C+S+I

★ Leave the first rating

Terms in this set (1564)

[1401] Which of these lines correctly prints 3?
 	
struct S {
int a = 3;
double b = 2.5;
};
 	
S obj, *p = &obj;
 	
cout << p.a << endl;
cout << *p.a << endl;
cout << *(p).a << endl;
cout << *(p.a) << endl;
cout << (*p).a << endl;
	cout << (*p).a << endl;

[1402] Which of these lines correctly prints 2.5?
 	
struct S {
int a = 3;
double b = 2.5;
};
 	
S obj, *p = &obj;
 	
cout << *(p).b << endl;
cout << *p.b << endl;
cout << p->b << endl;
cout << *(p.b) << endl;
cout << *p->b << endl;
	cout << p->b << endl;

[1403] Which of these lines displays the eighth element of a?
 	
int a[15];
 	
cout << a[8] << endl;
cout << a(7) << endl;
cout << a.at(7) << endl;
cout << a[7] << endl;
	cout << a[7] << endl;

[1404] Which prints the number of elements in a?
 	
int a[] = {1, 2, 3};
 	
cout << a.length << endl;
cout << sizeof(a[0]) << endl;
cout << a.size() << endl;
cout << sizeof(a) << endl;
None of these
	None of these

[1405] What is stored in the last element of nums? int nums[3] = {1, 2}; Undefined value 2 Syntax error in array declaration 0 1	0
[1406] Which line throws and out_of_range exception? double speed[5] = {. . .}; None of these cout << speed[4] << endl; cout << speed[5] << endl; cout << speed[0] << endl; cout << speed[1] << endl;	None of these

double speed[5] = {. . .};

cout << speed[5] << endl;
cout << speed[0] << endl;
None of these
cout << speed[1] << endl;
cout << speed[4] << endl;

[1408] Which line creates an array with 5 elements?

int[5] d;
int b[5];
int a[4];
None of these
int[] c[5];

int b[5];

[1409] What is printed?

int a[] = {1, 2, 3};
int b[] = {1, 2, 3};

if (a == b) cout << "a == b" << endl;
else cout << "a != b" << endl;

a != b
Undefined behavior
a == b
Syntax error; does not compile.

a != b

[1410] What does the array a contain after this runs?

int a[] = {1, 2, 3};
int b[] = {4, 5, 6};
a = b;

Syntax error; does not compile.
{4, 5, 6}
{1, 2, 3}
Undefined behavior

Syntax error; does not compile.

[1411] Which assigns a value to the first position in letters?

char letters[26];

letters[0] = 'a';
letters[0] = "a";
letters[1] = 'b';
letters.front() = 'a';
letters = 'a';

letters[0] = 'a';

[1412] Which assigns a value to the first position in letters?

char letters[26];

*letters = 'a';
*letters = "a";
*letters[0] = 'a';
*(letters + 1) = 'a';
*letters + 1 = 'b';

*letters = 'a';

[1413] What does this loop do?

int a[] = {6, 1, 9, 5, 1, 2, 3};
int x(0);
for (auto e : a) x += e;
cout << x << endl;

Counts the elements in a
Selects the largest value in a
Has no effect
Selects the smallest value in a
Sums the elements in a

Sums the elements in a

[1414] What is the address of the first pixel in the last row of this image?

Pixel *p; // address of pixel data
int w, h; // width and height of image

p + w + h
p + w + (h - 1)
p + w * h
p + w * (h - 1)
None of these are correct

p + w * (h - 1)

Pixel *p; // address of pixel data
int w, h; // width and height of image
 	
*p + w - 1
None of these are correct
*(p + w) - 1
p + w - 1
*(p + w - 1)	

[1416] Which returns the last pixel on the first row of this image?
 	
Pixel *p; // address of pixel data
int w, h; // width and height of image
 	
p[w - 1]
*p[w - 1]
None of these are correct
p[w] - 1
p + w - 1
	p[w - 1]

[1417] What is the equivalent array notation? int dates[10]; cout << (*dates + 2) + 2 << endl; dates[0] + 4 dates[2] + 2 dates[2] dates[0] + 2 &dates[2]	dates[0] + 4
[1418] What is the equivalent array notation? int dates[10]; cout << (dates + 2) << endl; dates[2] + 2 &dates[2] dates[0] + 2 dates[2] dates[0] + 4	&dates[2]
[1419] What is the equivalent array notation? int dates[10]; cout << *(dates + 2) << endl; dates[2] + 2 dates[0] + 4 dates[2] &dates[2] dates[0] + 2	dates[2]
[1420] What is the equivalent array notation? int dates[10]; cout << (*dates) + 2 << endl; &dates[2] dates[0] + 2 dates[0] + 4 dates[2] dates[2] + 2	dates[0] + 2
[1421] What is the equivalent array notation? int dates[10]; cout << *dates + 2 << endl; &dates[2] dates[2] + 2 dates[0] + 4 dates[2] dates[0] + 2	dates[0] + 2
[1422] What is the equivalent array notation? int dates[10]; cout << *(dates + 2) + 2 << endl; &dates[2] dates[0] + 4 dates[0] + 2 dates[2] dates[2] + 2	dates[2] + 2

int a[] = {1, 2, 3, 4, 5, 6, 7}; int *p = a; cout << a[1] * 2 << endl; None of these ***p + 1** * 2 p + 1 * 2 (***p + 1)** * 2 ***(p + 1)** * 2	

[1424] What prints?
 	
int a[] = {1, 3, 5, 7, 9};
int *p = a;
cout << *p++;
cout << *p << endl;
 	
13
None of these
33
22
12
	13

[1425] What prints?
 	
int a[] = {1, 3, 5, 7, 9};
int *p = a;
cout << *++p;
cout << *p << endl;
 	
33
13
None of these
22
12
	33

[1426] What prints?
 	
int a[] = {1, 3, 5, 7, 9};
int *p = a;
cout << ++*p;
cout << *p << endl;
 	
13
12
None of these
22
33
	22

[1427] Which pointer initialization is illegal?
 	
int a[] = {1, 3, 5, 7, 9};
int *p3 = &a[1];
None of these
int *p1 = a;
int *p4 = &a;
int *p2 = a + 3;
	int *p4 = &a;

[1428] Which expression returns the number of countries? string countries[] = {"Andorra", "Albania", . . . }; len(countries) countries.length sizeof(countries) * sizeof(countries[0]) sizeof(countries) None of these	None of these
[1429] Which expression returns the number of countries? string countries[] = {"Andorra", "Albania", . . . }; sizeof(countries) len(countries) sizeof(countries) / sizeof(string) None of these sizeof(countries) * sizeof(countries[0])	sizeof(countries) / sizeof(string)

string countries[] = {"Andorra", "Albania", . . . };

len(countries)
sizeof(countries) * sizeof(countries[0])
sizeof(countries)
None of these
sizeof(countries) / sizeof(countries[0])

[1431] Which array definition is illegal?

int SIZE = 3;
int a1[SIZE];
int a2[3];
int a3[3]{};
int a4[] = {1, 2, 3};
int a5[3] = {1, 2};

a2
a3
None of these
a1
a5

a1

[1432] Which array definition contains undefined values?

int SIZE = 3;
int a1[SIZE];
int a2[3];
int a3[3]{};
int a4[] = {1, 2, 3};
int a5[3] = {1, 2};

a3
a1
None of these
a5
a2

a2

[1433] Which array definition is initialized to all zeros?

int SIZE = 3;
int a1[SIZE];
int a2[3];
int a3[3]{};
int a4[] = {1, 2, 3};
int a5[3] = {1, 2};

a5
a2
None of these
a3
a1

a3

[1434] Which array definition produces {0, 1, 2}?

int SIZE = 3;
int a1[SIZE];
int a2[3];
int a3[3]{};
int a4[] = {1, 2, 3};
int a5[3] = {1, 2};

a5
a3
None of these
a2
a1

None of these

[1435] Which array definition is illegal?

const int SIZE = 3;
int a1[SIZE];
int a2[3];
int a3[3]{};
int a4[] = {1, 2, 3};
int a5[2] = {1, 2, 3};
a2
a5
a3
None of these
a1

a5

int SIZE = 3;
int a1[SIZE];
int a2[3];
int a3[3]{};
int a4[] = {1, 2, 3};
int a5[3] = {1, 2};

a3
a5
a2
a1
None of these

An incomplete type and a forward reference generally mean the same thing.

In C++ using == to compare one array to another is permitted (if meaningless).

You must use an integral constant or literal to specify the size of a built-in C++ array.

The reinterpret_cast instruction changes way that a pointer's indirect value is interpreted.

If p is a pointer to a structure, and the structure contains a data member x, you can access the data member by using the notation: (*p).x

C++ arrays have no support for bound-checking.

In C++ assigning one array to another is illegal

The allocated size of a built-in C++ array cannot be changed during runtime.

The size of the array is not stored along with its elements.

If img is a pointer to the first byte in an image loaded into memory, Pixel is a structure as defined in your textbook, you can create a Pixel pointer pointing to the image by writing:
Pixel **p = reinterpret_cast<Pixel** >(img);

The subscripts of a C++ array range from 0 to the array size - 1.

C++ arrays have no built-in functions for inserting and deleting.

A forward reference can be used when you want to use a pointer to a structure as a data member without first defining the entire structure.

The elements of a C++ array created in a function are allocated on the stack.

The elements of a C++ array created outside of a function are allocated in the static-storage area.

The elements of a C++ string array with no explicit initialization, created in a function will be set to the empty string.

Explicitly initializing an array like this: int a[3] = {1, 2, 3}; requires the size to be the same or larger than the number of elements supplied.

In C++ printing an array name prints the address of the first element in the array.

In C++ there is no separate array variable. The array name is a symbolic representation of the address of the first element in the array.

In C++ initializing an array with the contents of another is illegal.

In C++ initializing an array with the contents of another is illegal.

C++ arrays produce undefined results if you access an element outside the array.

Explicitly initializing an array like this: int a[] = {1, 2, 3}; works in all versions of C++.

True

You may use any kind of integral variable to specify the size of a built-in C++ array.	
The elements of a C++ string array with no explicit initialization, created in a function will be set to null.	
Explicitly initializing an array like this: int a[3] = {1, 2, 3}; requires the size to be the same or smaller than the number of elements supplied.	
In C++ using == to compare one array to another is illegal.	
The allocated size of a built-in C++ array may be changed during runtime	
If img is a pointer to the first byte in an image loaded into memory, Pixel is a structure as defined in your textbook, you can create a Pixel pointer pointing to the image by writing: Pixel **p = static_cast<Pixel** >(img);	
The reinterpret_cast instruction produces a temporary value by converting its argument.	
In C++ initializing an array with the contents of another is permitted.	
C++ arrays use bound-checking when you access their elements with the at() member function.	
The elements of a C++ array created in a function are allocated on the heap.	
In C++ assigning one array to another is permitted.	
C++ arrays throw an out_of_bounds exception if you access an element outside the array.	
In C++ an array variable and the array elements are separate. The array variable contains the address of the first element in the array.	
In C++ printing an array name prints the value of the first element in the array.	
The elements of a C++ int array with no explicit initialization, created in a function will be set to zero.	
C++ arrays can be allocated with a size of 0.	
The static_cast instruction changes way that a pointer's indirect value is interpreted.	
The size of the array is stored along with its elements.	
The allocated size of a built-in C++ array may be changed during runtime	
A forward reference can be used when you want to use a structure as a data member without first defining the entire structure.	
The elements of a C++ array created outside of a function are allocated on the stack.	
If p is a pointer to a structure, and the structure contains a data member x, you can access the data member by using the notation: *p->x	
C++ arrays offer built-in member functions for inserting and deleting.	
Explicitly initializing an array like this: int a[] = {1, 2, 3}; only works in C++ 11.	

Unix and C	Ken Thomson and Dennis Ritchie
Fortran	John Backus
Simula	O. Dahl & K. Nygaard
Berkeley Systems Distribution Unix	Bill Joy
C++	Bjarne Stroustrop
GNU, GCC and Free Software	Richard Stallman

Code is written in machine (and assembly) language for a specific processor; thus it is non-portable or machine dependent.	native code machine language

Which of these statements apply to C++?	More efficient than Java or Python Produces native code that runs on the CPU Compiles to native code

Converts processed source code to object code.	Compiler

Allows you to run your program in a controlled environment.	Debugger

Combines object modules to produce an executable.	Linker
Provides instructions for building your program.	Make
Reads an executable image on disk and starts it running.	Loader
Performs text substitution on your source code.	Preprocessor
What is wrong with this IPO code fragment? cout << "Name: "; string name; cout << "Hello, " << name << endl; cin >> name;	Input occurs after output
>> cout << cin \n endl	Extraction or input operator Analogous to Java's System.out Insertion or output operator Similar to Java's Scanner objects Escape character Stream manipulator
What kind of error is this? error: expected ';' after expression	A syntax error
What is the problem here? You have submitted another student's completion code	You filled out the STUDENT variable incorrectly
What is the problem here? make: *** No targets specified and no makefile found. Stop.	The programmer is in the wrong directory.

The makefile for h04 is missing
 	
int main()
{
 	
}
	Compiles, runs and returns 0 to the O/S

Below is the main function from the f2c program in Chapter 1. Which line(s) contain a function declaration?
 	
int main()
{
15 cout << "Enter a temperature in fahrenheit: ";
16 double fahr;
17 cin >> fahr;
18 double celsius = convert(fahr);
19 cout << "Converted: " << fahr << "F -> " << celsius << "C" << endl;
return 0;
}
	None of these

Below is the main function from the f2c program in Chapter 1. Which line(s) uses the character input stream?
 	
int main()
{
15 cout << "Enter a temperature in fahrenheit: ";
16 double fahr;
17 cin >> fahr;
18 double celsius = convert(fahr);
19 cout << "Converted: " << fahr << "F -> " << celsius << "C" << endl;
return 0;
}
	Line 17

Below is the main function from the f2c program in Chapter 1. Which line(s) contain a function call?
 	
int main()
{
15 cout << "Enter a temperature in fahrenheit: ";
16 double fahr;
17 cin >> fahr;
18 double celsius = convert(fahr);
19 cout << "Converted: " << fahr << "F -> " << celsius << "C" << endl;
return 0;
}
	Line 18

int main()
{
15 cout << "Enter a temperature in fahrenheit: ";
16 double fahr;
17 cin >> fahr;
18 double celsius = convert(fahr);
19 cout << "Converted: " << fahr << "F -> " << celsius << "C" << endl;
return 0;
}	

Below is the main function from the f2c program in Chapter 1. Which line(s) contain a output statement?
 	
int main()
{
15 cout << "Enter a temperature in fahrenheit: ";
16 double fahr;
17 cin >> fahr;
18 double celsius = convert(fahr);
19 cout << "Converted: " << fahr << "F -> " << celsius << "C" << endl;
return 0;
}
	Line 15
	Line 19

Below is the main function from the f2c program in Chapter 1. Which line(s) contain a variable defintion?
 	
int main()
{
15 cout << "Enter a temperature in fahrenheit: ";
16 double fahr;
17 cin >> fahr;
18 double celsius = convert(fahr);
19 cout << "Converted: " << fahr << "F -> " << celsius << "C" << endl;
return 0;
}
	Line 16
	line 18

Explain this output.
Why is nothing printed?
 	
#include <iostream>
using namespace std;
 	
int main()
{
cout << "Hello, World";
}
 	
make example
./example
	File not saved

What command only builds hw04?	make
What command checks hw04 for correctness?	make test
What command hands in hw04 for course credit?	make submit
What command makes hw the current folder?	cd ~/workspace/cs150/hw
1 cout << 10 + 1 << endl; 2 cout << (10 + 1) << endl; 3 (cout << 11) << endl;	Rule 1 precedence Rule 2 associativity Rule 3 side effect
The _____ of an operator determines which operands the operator binds with?	precedence
The _____ of an operator determines the order of operations when operators share an operand?	associativity
The _____ of an operator determines the number of items it operates on?	arity

What prints?
 	
int main()
{
cout << fixed << 2.0 / 3.0 << endl;
}
	.666667

What prints?
 	
int main()
{
cout << 2000 / 3.0 << endl;
}
	6.66667e-02

int main()
{
cout << fixed << 2000 / 3.0 << endl;
}	

What prints?
 	
int main()
{
cout << setprecision(2) << 2000 / 3.0 << endl;
}
	6.67e02

What prints?
 	
int main()
{
cout << fixed << setprecision(2) << 2000 / 3.0 << endl;
}
	666.67

Match each item with the correct statement below. unsigned long signed int unsigned long long unsigned int signed long signed long long	15UL 12 15ULL 15U 3L 15LL
The standard input object; analogous to a Scanner in Java Modifies and manipulates data to produce information The header used to include the standard streams A single entity that bundles data and instructions Displays the results of calculations cout stands for _____ The standard output object; analogous to System.out in Java The output or insertion operator endl is a _____ Retrieves data and stores it in variables C++ uses an _____ library for input and output. Text enclosed in double quotes The header used for formatting real numbers Asking an object to perform certain operations	cin processing iostream object output character output cout << stream manipulator input object-oriented string iomanip sending a message

Literals like 3 and 7 are always:
 	
On line 3, b is:
 	
int a = 3; // 1
int a = 7; // 2
a = b; // 3
	B O T H : rvalue

On line 2, b is:
 	
int main()
{
a = 3; // 1
const int b = 7; // 2
a = b; // 3
}
	A non-modifiable lvalue

Which operator is the extraction operator?	>>
Which operator is the insertion operator?	<<
What header is needed to use the string type?	<string>
What header is needed to use cin and cout?	<iostream>
What header is needed for output formatting?	<iomanip>
What header is needed to use the sqrt() function?	<cmath>

int n = 12;
cout << n/3 << endl; // 1
cout << n/7 << endl; // 2
cout << n % 3 << endl; // 3
cout << n % 7 << endl; // 4	

What is printed when this runs?
 	
int sum = 22;
sum +=2;
cout << sum++; // sum = sum + 4
cout << sum << endl;
	2425

What is the output of the following program?
int value = 3;
value++;
cout << value << endl;
	4

Which line or lines are illegal?
 	
/**1**/ int a, b;
/**2**/ a = 3;
 	
int main()
{
 	
/**3**/ b = 4;
/**4**/ cout << a << ", " << b << endl ;
 	
}
	2

Which line or lines are illegal?
 	
int a;
int b = 3;
 	
int main()
 	
{
a = 4;
cout << a << ", " << b << endl;
}
	None of these

Symbols which directly represent a value (literal) Determines the direction of operations for operators at the same level (associativity) Describes how many operands an operator requires(arity) Operators that require a single data value (unary) Symbol which indicates a value (operand) Determines how tightly operators bind to operands (precedence) Any combination of operators and operands which yields a value (expression) A symbol that can be used to produce a value at runtime (function call) Symbol which indicates an operation (operator) A storage location containing a value (variable) Operators that require two data values(binary)	literal associativity arity unary operand precedence expression function call operator variable binary
Types such as classes, structures and enumerations Types such as pointers, arrays and references Built-in types, such as int and double The "kind" of a variable Read a value and store it in a variable Types such as string and vector	user-defined types derived types primitive types data type input library types

Which of these five concepts are illustrated here?
 	
int main()
{
int a;
}
	Declaration
	Definition

int main()
{
extern int a;
}	

Which of these five concepts are illustrated here?
 	
int main()
{
extern int a;
a = 3;
}
	Assignment
	Declaration

Associates a name with a type Read a value and store it in a varaible Copy a new value into an existing variable Allocates space for a variable Provides a starting value when a variable is created A named storage area that holds a value	declare input assign define initialize variable
What is true about identifiers in C++?	They may contain an underscore
As an application programmer, which of the following names for local variables are both legal and recommended for stylistic reasons.	_ (single underscore) CamelCase
As an application programmer, which of the following names for local variables are legal (even if they are unwise from a stylistic perspective).	cout 2cool CamelCase u2 integer
Which manipulator is used to ensure that large numbers appear using regular decimal notation?	fixed
Which manipulator is used to change the padding character used in a column like: 0045?	setfill()
Which manipulator(s) is/are used to make sure the value 2.0/3 prints like this: 0.677?	fixed setprecision()
Which manipulator(s) is/are used to make sure the number 45 prints like this: 0045?	setfill() setw()

Assume int x, y, z;
 	
Shorthand assignment
 	
Post increment
 	
Undefined behavior
 	
Widening conversion
 	
Pre decrement
 	
Chained assignment
 	
Narrowing conversion
 	
Mixed-type expression
	y += z;
	
	x++;
	
	x = z++ - ++z;
	
	double a = y;
	
	--z;
	
	x = y = z = 10;
	
	z = 3.15;
	
	auto v = x * 2.3;

Which of the following variables have the value 0?
 	
int global;
int main
{
string localStr;
double localDouble;
}
	global

Which of the following variables have an undefined value?
 	
int global;
int main
{
string localStr;
double localDouble;
}
	localDouble

int global;
int main
{
string localStr;
double localDouble;
}	

The variable ASSIGNMENT from your homework has been _____.	declared
The variable STUDENT from your homework has been _____.	initialized defined declared

This code is legal, compiles and is well defined. Which line(s) contain an assignment?
 	
int a = 5; // 1
a == 5; // 2
int b = 6; // 3
a ={b}; // 4
auto c = a == b; // 5
	4

This code is legal, compiles and is well defined. Which line(s) contain comparison?
 	
int a = 5; // 1
a == 5; // 2
int b = 6; // 3
a ={b}; // 4
auto c = a == b; // 5
	2
	5

This code is legal, compiles and is well defined. Which line(s) contain initialization?
 	
int a = 5; // 1
a == 5; // 2
int b = 6; // 3
a ={b}; // 4
auto c = a == b; // 5
	1
	3
	5

This code is legal, compiles and is well defined. Which line(s) contain an input statement?
 	
int a = 5; // 1
a == 5; // 2
int b = 6; // 3
a ={b}; // 4
auto c = a == b; // 5
	None of these

The + arithmetic operator is a(n) _____ operator	binary
The - operator is a(n) _____ operator	unary binary
The ++ arithmetic operator is a(n) _____ operator	side effect unary
A set of bits interpreted according to its type	Value
x in the expression x = 3;	lvalue
x in the expression y = x;	rvalue
Uniform or list initialization	int c{5};
Legacy or assignment initialization	int a = 0;
Direct initialization	int b(3);
const double PI = 3.14159;	non-modifiable value
narrowing conversion	int e(3.5);
Assume that the user enters: Steve 60 3.5 What value is stored in gpa? string name; int age; double gpa; cout << "Enter your name, age and gpa: "; cin >> name >> age >> gpa;	3.5

```
string name;
int age;
double gpa;

cout << "Enter your name, age and gpa: ";
cin >> name >> age >> gpa;
```

Assume that the user enters: Steve 3.5 68
What value is stored in age?

```
string name;
int age;
double gpa;

cout << "Enter your name, age and gpa: ";
cin >> name >> age >> gpa;
```

3

Assume that the user enters: Steve Gilbert 68 3.5
What value is stored in age?

```
string name;
int age;
double gpa;

cout << "Enter your name, age and gpa: ";
cin >> name >> age >> gpa;
```

undefined

Which of these are impossible conditions?

```
auto floor ??? // some number;

bool v1 = floor >= 0 || floor <= 20;
bool v2 = floor <= 0 && floor >= 20;
bool v3 = floor <= 0 || floor >= 20;
bool v4 = floor >= 0 && floor <= 20;
bool v5 = floor >= 0 || floor < 20;
bool v6 = floor >= 0 && floor > 20;
bool v7 = floor >= 0 || floor > 20;
bool v8 = floor >= 0 && floor < 20;
```

v2

Which of these are unavoidable conditions?

```
auto floor ??? // some number;

bool v1 = floor >= 0 || floor <= 20;
bool v2 = floor <= 0 && floor >= 20;
bool v3 = floor <= 0 || floor >= 20;
bool v4 = floor >= 0 && floor <= 20;
bool v5 = floor >= 0 || floor < 20;
bool v6 = floor >= 0 && floor > 20;
bool v7 = floor >= 0 || floor > 20;
bool v8 = floor >= 0 && floor < 20;
```

v1
v5

[] and () denote whether a range includes or excludes an endpoint:
[includes the endpoint
(excludes the endpoint
[] = 'Closed', includes both endpoints
() = 'Open', excludes both endpoints
[) and (] are both 'half-open', and include only one endpoint

Which variable correctly indicates that the variable floor is in the range [0...20)?

```
auto floor ??? // some number;
bool v1 = floor >= 0 || floor <= 20;
bool v2 = floor <= 0 && floor >= 20;
bool v3 = floor <= 0 || floor >= 20;
bool v4 = floor >= 0 && floor <= 20;
bool v5 = floor >= 0 || floor < 20;
bool v6 = floor >= 0 && floor > 20;
bool v7 = floor >= 0 || floor > 20;
bool v8 = floor >= 0 && floor < 20;
```

v8

(excludes the endpoint
[] = 'Closed', includes both endpoints
() = 'Open', excludes both endpoints
[) and (] are both 'half-open', and include only one endpoint
 	
Which variable correctly indicates that the variable floor is in the range (0...20)?
 	
auto floor ??? // some number;
bool v1 = floor >= 0 || floor <= 20;
bool v2 = floor <= 0 && floor >= 20;
bool v3 = floor <= 0 || floor >= 20;
bool v4 = floor >= 0 && floor <= 20;
bool v5 = floor >= 0 || floor < 20;
bool v6 = floor >= 0 && floor > 20;
bool v7 = floor >= 0 || floor > 20;
bool v8 = floor >= 0 && floor < 20;	

[] and () denote whether a range includes or excludes an endpoint:
[includes the endpoint
(excludes the endpoint
[] = 'Closed', includes both endpoints
() = 'Open', excludes both endpoints
[) and (] are both 'half-open', and include only one endpoint
 	
Which variable correctly indicates that the variable floor is in the range [0...20]?
 	
auto floor ??? // some number;
bool v1 = floor >= 0 || floor <= 20;
bool v2 = floor <= 0 && floor >= 20;
bool v3 = floor <= 0 || floor >= 20;
bool v4 = floor >= 0 && floor <= 20;
bool v5 = floor >= 0 || floor < 20;
bool v6 = floor >= 0 && floor > 20;
bool v7 = floor >= 0 || floor > 20;
bool v8 = floor >= 0 && floor < 20;
	v4

Strings in C++ are mutable. String in C++ are immutable	True False
In C++ you can compare strings using all of the relational operators. In C++ you cannot use the relational or equality operators with strings.	True False
Assuming str is a string object, this syntax is legal in both Java and C++. Does this code work correctly in both languages? if (str == "quit") . . .	False
In C++ you can concatenate string objects using the + or += operators.	True
Assuming str is a string object, is this syntax legal in both Java and C++? if (str == "quit") . . .	True
Assuming str is a string object does this correctly test if str consists of the characters "quit" in C++? if (str == "quit") . . .	True
Assuming lastName is a string object, does this work as expected in C++? if (lastName <= "Gilbert") . . .	True
fruitful function	A function that calculates and returns a value
body	A block containing statements that implement the function's actions.
function	A named block of code that carries out an action or calculates a value.
prototype	Another name for a function declaration
parameters	Variables defined along with the function to receive input
calling	Executing, running or invoking the function
procedure	A function that carries out an action instead of calculating a value.
return statement	Produces a value when the function is invoked
defining	Specifying the calculation or actions that occur when the function is used
declaring	Specifying the function name, type and parameter types.
arguments	Values passed to the function when it is invoked

Which control structure is best equipped to handle an on or off condition?	if-else statements
Which control structure is best equipped to handle numeric selections made from a menu?	the switch statement
Which control structure is best equipped to handle processing for a group of radio buttons?	sequential if statements
Which control structure is best equipped to handle processing for income taxes?	nested if statements
Which control structure is best equipped to set a variable to one or two possible values?	the conditional operator
This code illustrates the _____ idiom. if (n % 2 == 1) cout << "Odd" << endl; else cout << "Even" << endl;	alternative action
This code illustrates the _____ idiom. if (n % 2 == 1) cout << "Odd" << endl;	guarded action

This code illustrates the _____ idiom.
 	
auto n = 3;
if (n % 2 == 1) n = -n;
else if (n < 0) n++;
else if (n % 2 = 0) n--;
else n = 0;
	multiple selection

This code illustrates the ____ idiom.
 	
auto n = 3;
if (n % 2 == 1) n = -n;
if (n < 0) n++;
if (n % 2 = 0) n--;
	Independent if statements

This code illustrates the ____ idiom.
 	
auto n = 3;
else if (n % 2 == 1) n = -n;
else if (n < 0) n++;
else if (n % 2 = 0) n--;
else n = 0;
	None of these are correct

The C++ string class is defined in the header:	<string>
You can find the length of a string str using str.size(). In C++, size() is called:	a member function
Which operator is used to see if all of a set of conditions is true?	logical and
Which operator is used to see if any of a set of conditions is true?	logical or
If a is false, which expressions need not be evaluated? if (a && b || c && d || e) . . .	b
If a and c are both false, which expressions need not be evaluated? if (a && b || c && d || e) . . .	b, d
If a and b are true, which expressions need not be evaluated? if (a && b || c && d || e) . . .	c, d, e
Produces the empty string	string s1; (choice A)
Implicitly converts a character array to a string object	string s2 = "hello"; (choice B)
Explicitly converts a character array to a string object	string s3{"world"}; (choice C)
Produces a string from multiple copies of a single character	string s4(20, '-'); (choice D)
Produces a string that may contain quotes or backslashes	string s5(R"("bob")"); (choice E)
Needed to use the C++ string type	#include <string> (choice F)
Reads one word or token from standard input	cin >> s1; (choice G)
Reads one line of text from standard input	getline(cin, s2); (choice H)

cout << "What's your name: "; string name; cin >> name; cout <<"Howdy " + name + "!";	

What is the output?
 	
auto x = 40;
if (x <= 40) cout << "F";
if (x <= 75) cout << "C";
if (x <= 90) cout << "B";
cout << endl;
	FCB

In C++, what is true about concatenating string literals (character arrays)?	you do it by separating the string literals with white space
In C++, the statement string s{3, 'X'};	creates a string variable of size 3, filled with 'X'
In C++, the statement string s{"world"};	creates a string variable explicitly initialized with the character array "world"
In C++, the statement string s = "world";	creates a string variable implicitly initialized with the character array "world"
In C++, what keyword is used for type inference?	auto
In C++, characters of the type char:	Can be preceded by signed or unsigned for use as small integers Generally use 8 bits of storage. Use the ASCII character set Are defined for the first 127 characters
In C++, characters of type char:	Can be preceded by signed or unsigned for use as small integers Generally use 8 bits of storage. Use the ASCII character set Are defined for the first 127 characters
In C++, what is true about the += operator operating on string objects	You may concatenate creates a string variable to a string object You may concatenate a string literal (character array) to a string object You may concatenate a char literal to a string object You may concatenate a char variable to a string object

The code shown here:
 	
auto n = 3;
if (n = 0)
cout << "n is 0" << endl;
else
cout << "n is " << n << endl;
	Executes the false branch
	Displays "n is 0"
	Contains an embedded assignment

Assume that name is a string object. Which of these expressions are legal?	name += 'X' name < "bob" name == "sally" name += "fred"
What is true about string::size_type?	It is the same as size_t It is returned from the string size() member function It is returned from the string length() member function It is an unsigned integer type of some size You may create variables of that type
Compare C++ and Java string. Which of these are true?	"hello" is a string object in Java, but not in C++ String s; produces the null string in Java, while string s; produces the empty string in C++. String is capitalized in Java, lowercase in C++ Assuming str is a string, str + "b" is legal in both Java and C++
What header file do you include to call the isupper() function?	<cctype>
All of these are declared in the <string> header; which are member functions?	size() front() find() at()

Match the letter of the variable in the figure with the correct value or expression below
 	
string s{"walk the plank"};
 	
auto a = s.find('a');
auto b = s.find('a', 3);
auto c = s.find("nk");
auto d = s.find("Walk");
	a : 1
	b : 11
	c : 12
	d : string::npos

One-way, independent decisions use:	if
Either/or decisions should use:	if . . . else
Multiple possible outputs, testing a single condition, use:	if . . . else . . . if . . . else
Leveled decisions, such as processing income taxes are best handled with:	if . . . if . . . else . . . else

To combine several test conditions to produce a single Boolean value, use:	a logical operator
In Line 2, what is the receiver? string s{"happy"}; auto pos = s.find('y');	s
Decisions based on numbered blocks of code are best handled with:	switch
In Line 2, what is the result of this function call? string s{"happy"}; auto pos = s.find('y');	pos
In Line 2, what is the request? string s{"happy"}; auto pos = s.find('y');	find
In Line 2, what is the explicit argument? string s{"happy"}; auto pos = s.find('y');	'y'
In Line 2, what is the implicit argument? string s{"happy"}; auto pos = s.find('y');	the address of s
In Line 2, what is the parameter? string s{"happy"}; auto pos = s.find('y');	None of these

Assume c is a char variable. What type is the variable a?
 	
string s{"guten tag"};
auto len = s.size();
auto a = s.front();
s.at(len) = a;
s[len] = c;
	char

Assume c is a char variable. What value s stored in the variable a?
 	
string s{"guten tag"};
auto len = s.size();
auto a = s.front();
s.at(len) = a;
s[len] = c;
	'g'

What type is the variable len?
 	
string s{"guten tag"};
auto len = s.size();
auto a = s.front();
s.at(len) = a;
s[len] = c;
	string::size_type

Assume c is a char variable. What type is the expression s.last()?
 	
string s{"guten tag"};
auto len = s.size();
auto a = s.front();
s.at(len) = a;
s[len] = c;
	None of these

The relative order of two variables is tested using:	a relational operator

Assume c is a char variable. Which line throws an error because of range checking?
 	
string s{"guten tag"}; // 1
auto len = s.size(); // 2
auto a = s.front(); // 3
s.at(len) = a; // 4
s[len] = c; // 5
	4

Assume c is a char variable. Which line produces undefined behavior?
 	
string s{"guten tag"}; // 1
auto len = s.size(); // 2
auto a = s.front(); // 3
s.at(len) = a; // 4
s[len] = c; // 5
	5

string s{"guten tag"}; // 1
auto len = s.size(); // 2
auto a = s.front(); // 3
s.at(len) = a; // 4
s[len] = c; // 5	

The string find() member function may be used to search for a substring The string find() member function takes either a string or character as an argument The string find() member function throws an exception if the target cannot be found.	True True False
Calling s.at(1) returns a reference to the second character in the string object s Calling s.at(1) returns a copy of the second character in the string object s	True False
s.at(0) = 'c'; changes the first character in the string object s to 'c' s.at(0) = "c"; changes the first character in the string object s to 'c'	True False
The getline() function is part of the string class.	False
Data member is the term used in C++ for what is called a method in Java	False
The toupper() member function ignores case when it searches.	False
In C++ a char may be one, two or three bytes, when using UTF-8.	False
s.back() = 'x'; changes the last character in the string object s to 'x'.	True
Calling s.at(0) returns the same reference as s.front().	True
A C++ string that contains Unicode characters should be preceded by:	u8
To enter a Unicode character into a C++ string, use an escape sequence starting with:	\U
Which of these selects a character (char) from a string?	auto a = s[0];
This compiles, runs and prints 12. What is the correct parameter declaration for x?	int& x

This compiles, runs and prints 4, 3. What is the correct prototype?
 	
int x = 3, y = 4;
swap(x, y);
cout << x << ", " << y << endl;
	void swap(int& a, int& b);

What value is stored in a after this runs? string s{"ABCDEFD"}; auto a = s.find('D');	3
What value is stored in a after this runs? string s{"abcdefg"}; auto a = s.substr(3);	"defg"
What value is stored in a after this runs? string s{"ABC"}; auto a = s.substr(4, 5);	Runtime error because start (4) must bet 0..3
What value is stored in a after this runs? string s{"ABCDEFD"}; auto a = s.find('G');	string::npos
What value is stored in a after this runs? string s{"ABCDEFGHIJKLM"}; auto a = s.substr(1, 4);	"BCDE"
String parameters should be passed to functions:	by constant reference (const string& s) when not modified in the function. by reference (string& s) when modified in the function
Assume a is 5 and b is 3; what prints? string s{"feed the fish"}; cout << s.substr(a, b) << endl;	"the"

string s{"feed the fish"}; cout << s.substr(a, b) << endl;	
Assume a is 13 and b is 10; what prints? string s{"feed the fish"}; cout << s.substr(a, b) << endl;	""
Assume a is 14 and b is 10; what prints? string s{"feed the fish"}; cout << s.substr(a, b) << endl;	Runtime error

How many variables appear in the following code segment?
 	
int n = 4;
int& r1 = n;
auto& r2 = r1;
r1 = 3;
r2 = 5;
cout << n << endl;
	1

What does this code segment print?
 	
int n = 4;
int& r1 = n;
auto& r2 = r1;
r1 = 3;
r2 = 5;
cout << n << endl;
	5

Which of these lines are illegal? [1] int n1 = 4; [2] double n2 = 3.145; [3] unsigned char n3 = 158; [4] int n4 = n2; [5] int& r1 = n1; [6] int& r2 = n2; [7] double& r3 = n1; [8] const int& r4 = n2;	6 7
Which of these lines are legal? [1] int n1 = 4; [2] double n2 = 3.145; [3] unsigned char n3 = 158; [4] int n4 = n2; [5] int& r1 = n1; [6] int& r2 = n2; [7] double& r3 = n1; [8] const int& r4 = n2;	4 5 8
Which lines cause runtime errors (exceptions)? [1] string s{"shiver me timbers"}; [2] auto len = s.size(); [3] s.front() = 'S'; [4] s.back() = "S"; [5] s[len] = 'X'; [6] s.substr(0, 1) = "W"; [7] auto a = s.substr(0, 100); [8] auto b = s.substr(4, 3); [9] auto c = s.substr(len);	None of these
Which lines compile and return string objects? [1] string s{"shiver me timbers"}; [2] auto len = s.size(); [3] s.front() = 'S'; [4] s.back() = "S"; [5] s[len] = 'X'; [6] s.substr(0, 1) = "W"; [7] auto a = s.substr(0, 100); [8] auto b = s.substr(4, 3); [9] auto c = s.substr(len);	7 8 9
Which lines cause syntax errors? [1] string s{"shiver me timbers"}; [2] auto len = s.size(); [3] s.front() = 'S'; [4] s.back() = "S"; [5] s[len] = 'X'; [6] s.substr(0, 1) = "W"; [7] auto a = s.substr(0, 100); [8] auto b = s.substr(4, 3); [9] auto c = s.substr(len);	4 6

string s{"xyzw"}; s.at(2) = 'Y';	

string s{"ahoy"};
 	
auto a = s.size();
auto b = s.back();
auto c = s.at(0);
auto d = s.substr(a);
auto e = s.substr(0, 1);
	[a] : string::size_type
	[b] : 'y'
	[c] : 'a'
	[d] : ""
	[e] : "a"

[1] What must I change in the test to go to the next iteration? [2] What information is produced? [3] What must I do to enter the loop? [4] Can my loop reach its bounds? [5] Has my loop reached its goal? [6] How is the data processed? [7] Can my loop be entered at all? [8] What makes this loop quit?	[1] advance the loop [2] goal precondition [3] bounds precondition [4] necessary bounds [5] loop postcondition [6] loop operations and actions [7] loop guards [8] loop bounds
[1] May not repeat its actions at all [2] Keeps processing input until a particular value is found in input. [3] Repeats its actions at least once [4] Keeps processing until the output gets no closer to the answer. [5] Test for the occurrence of a particular event [6] Repeats its actions a fixed number of times [7] Conditions under which a loop will repeat its actions [8] Keeps processing until the input device signals that it is finished.	[1] guarded loop [2] sentinel loop [3] unguarded loop [4] limit loop [5] indefinite loop [6] definite loop [7] loop bounds [8] data loop
[1] Actions that occur after the loop is complete [2] Actions occuring inside the loop's body [3] Actions that occur before the loop is encountered [4] A test that determines if the loop should be entered	[1] postcondition [2] operation [3] precondition [4] bounds

Which of these is a flow-of-control statement?	for (auto e : s) ...
	
	if (x < 3) ... else ...
	
	while (x < 3) ...

Which of these are guarded loops?	for while
Which of these are unguarded loops?	do-while
Which are the two major categories of loops?	definite indefinite
Which of these are indefinite loops?	sentinel bounds limit bounds data bounds
Using the loop-building strategy from Chapter 5, which of these are part of the loop mechanics?	loop bounds bounds precondition advancing the loop
Look at the problem statement below. The _____ of the loop is to count the number of characters in a sentence. [How many characters are in a sentence? Count the characters in a string until a period is encountered. If the string contains any characters, then it will contain a period. Count the period as well.]	goal
Look at the problem statement below. The _____ of the loop is that a period was encountered. [How many characters are in a sentence? Count the characters in a string until a period is encountered. If the string contains any characters, then it will contain a period. Count the period as well.]	bounds
Look at the problem statement below. The _____ of the loop is read a character and increment a counter. [How many characters are in a sentence? Count the characters in a string until a period is encountered. If the string contains any characters, then it will contain a period. Count the period as well.]	plan
Loop bounds used when searching through input.	sentinel bounds
Loop bounds often used in scientific and mathematical applications.	limit bounds
In the classic for loop, loop control variables going from 0 to less-than n are said to employ:	asymmetic bounds
Loop bounds used when reading files or processing network data.	data bounds

for (int i = 1; i < 10; i++) cout << i; cout << endl;	
How many times is this loop entered? (That is, how many times is i printed?) for (int i = 1; i <= 10; i++) cout << i; cout << endl;	10
How many times is this loop entered? (That is, how many times is i printed?) for (int i = 0; i < 10; i++) cout << i; cout << endl;	10
How many times is this loop entered? (That is, how many times is i printed?) for (int i = 0; i <= 10; i++) cout << i; cout << endl;	11
In the classic for loop, which portion of code is not followed by a semicolon?	update expression
In the classic for loop, which portion of code is executed after the last statement in the loop body?	update expression
In the classic for loop, which portion of code is analogous to an if statement?	condition expression
In the classic for loop, which portion is used to create the loop control variable?	initialization statement

Below is the illustration from the loop building strategy in Chapter 5. The highlighted lines represents:
 	
Given: the variable str is a string (may be empty)
Create the counter variable, initialized to -1
 	
☼
If the variable str has any characters then
☼
 	
{
 	
Set counter to 0
Create the variable current-character as a character
Place the first character in str into current-character
 	
While more-characters and current-character not a period
{
Add one to (or increment) the counter variable
Store the next character from str in current-character
}
 	
If current-character is a period then
Add one to the counter to account for the period.
Else
Set counter to -2
 	
}
 	
If counter is -1 the string was empty
Else if counter is -2 there was no period
	a loop guard

Given: the variable str is a string (may be empty)
Create the counter variable, initialized to -1

If the variable str has any characters then
{

☼
Set counter to 0
☼

Create the variable current-character as a character
Place the first character in str into current-character

While more-characters and current-character not a
period
{
Add one to (or increment) the counter variable
Store the next character from str in current-character
}

If current-character is a period then
Add one to the counter to account for the period.
Else
Set counter to -2

}

If counter is -1 the string was empty
Else if counter is -2 there was no period

Below is the illustration from the loop building strategy in Chapter 5. The highlighted lines represents:

Given: the variable str is a string (may be empty)
Create the counter variable, initialized to -1

If the variable str has any characters then
{
Set counter to 0

☼
Create the variable current-character as a character
Place the first character in str into current-character
☼

While more-characters and current-character not a period
{
Add one to (or increment) the counter variable
Store the next character from str in current-character
}

If current-character is a period then
Add one to the counter to account for the period.
Else
Set counter to -2

}

If counter is -1 the string was empty
Else if counter is -2 there was no period

bounds precondition

Given: the variable str is a string (may be empty)
Create the counter variable, initialized to -1

If the variable str has any characters then
{

Set counter to 0
Create the variable current-character as a character
Place the first character in str into current-character

☼
While more-characters
☼

and current-character not a period
{
Add one to (or increment) the counter variable
Store the next character from str in current-character
}

If current-character is a period then
Add one to the counter to account for the period.
Else
Set counter to -2

}

If counter is -1 the string was empty
Else if counter is -2 there was no period

Below is the illustration from the loop building strategy in Chapter 5. The highlighted lines represents:

Given: the variable str is a string (may be empty)
Create the counter variable, initialized to -1

If the variable str has any characters then
{
Set counter to 0

Create the variable current-character as a character
Place the first character in str into current-character

☼
While more-characters and current-character not a period
☼

{
Add one to (or increment) the counter variable
Store the next character from str in current-character
}

If current-character is a period then
Add one to the counter to account for the period.
Else
Set counter to -2

}

If counter is -1 the string was empty
Else if counter is -2 there was no period

loop bounds

Given: the variable str is a string (may be empty)
Create the counter variable, initialized to -1

If the variable str has any characters then
{
Set counter to 0
Create the variable current-character as a character
Place the first character in str into current-character

While more-characters and
☼
current-character not a period
☼
{
Add one to (or increment) the counter variable
Store the next character from str in current-character
}

If current-character is a period then
Add one to the counter to account for the period.
Else
Set counter to -2

}

If counter is -1 the string was empty
Else if counter is -2 there was no period

Below is the illustration from the loop building strategy in Chapter 5. The highlighted lines represents:

Given: the variable str is a string (may be empty)
Create the counter variable, initialized to -1

If the variable str has any characters then
{
Set counter to 0
Create the variable current-character as a character
Place the first character in str into current-character

While more-characters and current-character not a period
{

☼
Add one to (or increment) the counter variable
☼

Store the next character from str in current-character
}
If current-character is a period then
Add one to the counter to account for the period.
Else
Set counter to -2

}

If counter is -1 the string was empty
Else if counter is -2 there was no period

goal operation

Given: the variable str is a string (may be empty)
Create the counter variable, initialized to -1
 	
If the variable str has any characters then
{
 	
Set counter to 0
Create the variable current-character as a character
Place the first character in str into current-character
 	
While more-characters and current-character not a period
{
Add one to (or increment) the counter variable
 	
☼
Store the next character from str in current-character
☼
 	
}
 	
If current-character is a period then
Add one to the counter to account for the period.
Else
Set counter to -2
 	
}
 	
If counter is -1 the string was empty
Else if counter is -2 there was no period	

Below is the illustration from the loop building strategy in Chapter 5. The highlighted lines represents:
 	
Given: the variable str is a string (may be empty)
Create the counter variable, initialized to -1
 	
If the variable str has any characters then
{
Set counter to 0
Create the variable current-character as a character
Place the first character in str into current-character
 	
While more-characters and current-character not a period
{
Add one to (or increment) the counter variable
Store the next character from str in current-character
}
 	
☼
If current-character is a period then
☼
 	
Add one to the counter to account for the period.
Else
Set counter to -2]
 	
}
 	
If counter is -1 the string was empty
Else if counter is -2 there was no period
	loop postcondition

In a guarded loop, the loop actions may never be executed In a guarded loop, the loop actions are always executed at least once.	True False
In an unguarded loop, the loop actions are always executed at least once. In an unguarded loop, the loop actions may never be executed.	True False
A guarded loop is also known as a test-at-the-top loop A guarded loop is also known as a test-at-the-bottom loop.	True False
An unguarded loop is also known as a test-at-the-bottom loop. An unguarded loop is also known as a test-at-the-top loop.	True False
Loops are used to implement iteration in C++. Loops are used to implement selection in C++.	True False

for (int i = 1; i <= 10; i++)
cout << i;
cout << endl;

This idiomatic pattern is used to count from one value to another.

for (int i = 1; i < 10; i++)
cout << i;
cout << endl;

False

This loop uses asymmetric bounds.

for (int i = 0; i < 10; i++)
cout << i;
cout << endl;

True

This loop uses asymmetric bounds.

for (int i = 1; i < 10; i++)
cout << i;
cout << endl;

True

This loop uses asymmetric bounds.

for (int i = 1; i <= 10; i++)
cout << i;
cout << endl;

False

What prints?

```
void fn(int, double, double&) { cout << "A" << endl; }
void fn(int, int, double&) { cout << "B" << endl; }
void fn(int, int, double) { cout << "C" << endl; }
void fn(int, int, int) { cout << "D" << endl; }

int main()
{
auto n = 3.5;
fn(1, 2.5, n);
}
```

A

What prints?

```
void fn(int, double, double&) { cout << "A" << endl; }
void fn(int, int, double&) { cout << "B" << endl; }
void fn(int, int, double) { cout << "C" << endl; }
void fn(int, int, int) { cout << "D" << endl; }

int main()
{
fn(2.5, 1.5, 2.5);
}
```

C

What prints?

```
void fn(int, double, double&) { cout << "A" << endl; }
void fn(int, int, double&) { cout << "B" << endl; }
void fn(int, int, double) { cout << "C" << endl; }
void fn(int, int, int) { cout << "D" << endl; }

int main()
{
fn(1, 2, 3.5);
}
```

C

What prints?

```
void fn(int, double, double&) { cout << "A" << endl; }
void fn(int, int, double&) { cout << "B" << endl; }
void fn(int, int, double) { cout << "C" << endl; }
void fn(int, int, int) { cout << "D" << endl; }

int main()
{
fn(2.5, 1.5, 7);
}
```

D

void fn(int, double, double&) { cout << "A" << endl; }
void fn(int, int, double&) { cout << "B" << endl; }
void fn(int, int, double) { cout << "C" << endl; }
void fn(int, int, int) { cout << "D" << endl; }
 	
int main()
{
fn(1, 2, 3, 4);
}	

What prints?
 	
void fn(int, double, double&) { cout << "A" << endl; }
void fn(int, int, double&) { cout << "B" << endl; }
void fn(int, int, double) { cout << "C" << endl; }
void fn(int, int, int) { cout << "D" << endl; }
 	
int main()
{
auto n = 3.5;
fn(1, 2, n);
}
	Syntax error: ambiguous

What prints here?
 	
auto a = 3, b = 3;
cout << (a != b ? "panda": "tiger") << endl;
	tiger

What prints here?
 	
auto a = 4, b = 3;
cout << (a == b ? "panda": a % 2 ? "stork": "tiger") << endl;
	tiger

What prints here?
 	
auto a = 3, b = 3;
cout << (a == b ? "panda": "tiger") << endl;
	panda

What prints here?
 	
auto a = 3, b = 3;
cout << (a != b ? "panda": a % 2 ? "stork": "tiger") << endl;
	stork

What prints here?
 	
auto a = 3, b = 3;
cout << a == b ? "panda" : "tiger" << endl;
	Does not compile

Function overloading allows you to write several different functions that have the same name. Function overloading lets you call a single function in several different ways.	True False
Overloaded functions have the same name but different parameter types. Overloaded functions have the same name but different parameter names.	True False
In a while loop, (condition) is followed by a semicolon. A while loop is a hasty or unguarded loop.	False False

What prints here?
 	
auto a = 1;
switch (a)
{
case 1: cout << "1"; break;
case 2: cout << "2"; break;
default: cout << "3";
}
cout << endl;
	1

What prints here?
 	
auto a = 2;
switch (a)
{
case 1: cout << "1"; break;
case 2: cout << "2"; break;
default: cout << "3";
}
cout << endl;
	2

auto a = '1';
switch (a)
{
case 1: cout << "1"; break;
case 2: cout << "2"; break;
default: cout << "3";
}
cout << endl;	

What prints here?
 	
auto a = 1;
switch (a)
{
case 1: cout << "1";
case 2: cout << "2";
}
cout << endl;
	12

What prints here?
 	
auto a = 1;
switch (a)
{
case 1: cout << "1";
case 2: cout << "2";
case 3:
}
cout << endl;
	Does not compile

What prints here?
 	
double a = 1;
switch (a)
{
case 1: cout << "1";
case 2: cout << "2";
}
cout << endl;
	Undefined behavior

What prints here?
 	
auto a = 'A';
switch (a)
{
case 64: cout << "?";
case 65: cout << "A";
case 66: cout << "B";
}
cout << endl;
	A
	But should be AB

The compiler determines which overloaded function to call by looking at the number, types and order of the arguments passed to the function.	True
Default arguments let you call a single function in several different ways. Default arguments allow you to write several different functions that have the same name.	True False
Default arguments may only be used with value parameters. Default arguments may only be used with reference parameters. Default arguments may be used with both value and reference parameters.	True False False
Default arguments appear only in the function prototype. Default arguments appear only in the function implementation.	True False
Fatal error messages should be printed to cerr. Fatal error messages should be printed to cout.	True False
Calling break() terminates a program immediately and passes an error code back to the operating system.	False
The compiler determines which overloaded function to call by looking at the type of value the function returns.	False
If str = "hello", then str.size() > -1.	False
Calling exit() terminates a program immediately and passes an error code back to the operating system.	True

A do-while loop is also called a hasty loop.	True
In a do-while loop, (condition) is followed by a semicolon.	True

To allow f() to change the argument passed here, the parameter str should be declared as:
 	
void f(. . . str);
int main()
{
string s = "hello";
f(s);
}
	string&

To allow f() to accept the argument passed here, the parameter str should be declared as:
 	
void f(. . . str);
int main()
{
f("hello");
}
	const string&

To allow f() to change the argument passed here, the parameter str should be declared as:
 	
void f(. . . str);
int main()
{
f("hello");
}
	It is not possible for f() to change the argument passed here.

A function where an argument is converted to match a parameter When more than one match is found for the proffered arguments. A function where each argument is the same type as the corresponding parameter. A group of functions with the same name. A group of functions that have the same name and the correct number of parameters. When no match is found for the proffered arguments	best match ambiguity exact matches candidate set viable set empty set

Examine the following variables and function calls
Match each item with the correct statement below.
int able = 3;
int baker = f1(able);
cout << able << baker << endl; // 64
 	
int charlie;
f2("hello", charlie);
cout << charlie << endl; // Hello Carl
	Returned value --> baker
	
	Output argument (parameter) --> Charlie
	
	Input argument (parameter) --> Hello
	
	Input/output argument (parameter) --> able

Which of these are not ways that functions may be overloaded?	different function name different return type different parameter names
Different functions that have the same name, but take different arguments, are said to be:	overloaded
You can call a single function in several different ways by giving the function _____:	default arguments

Given the overloaded functions prototypes and the variable definition below, which of the function calls will fail to compile?
 	
int f(int&);
int f(int);
int f(int, int);
int a = 7;
	f(a);

Given the overloaded functions prototypes and the variable definition below, which of the function calls will fail to compile?
 	
int f(int&);
int f(const int&);
int f(int, int);
int a = 7;
	None of these fail to compile

int i = 1;
int n;
cin >> n;
do
{
i++;
cin >> n;
}
while (n % 2);
cout << i << endl;	

Assume that the input is 5 5 4 3 5. What will print?
 	
int i = 1;
int n;
do
{
cin >> n;
i++;
}
while (n % 2);
cout << i << endl;
	4

int i = 1;
int n;
do
{
cin >> n;
i++;
}
while (n % 2);
cout << i << endl;
	lvalues

Examine this code. Which is the best prototype? int age; string name = read("Enter your name, age: ", age);	string read(const string&, int&)

What prints?
 	
string str = "Hello";
for (int i = str.size() - 1; i >= 0; i--)
cout << str.at(i);
	olleH

What prints?
 	
string str = "Hello";
for (size_t i = str.size() - 1; i >= 0; i--)
cout << str.at(i);
	Crashes when run

What prints?
 	
string str = "Hello";
for (auto i = 0, len = str.size(); i < len; i++)
cout << str.at(i);
	Does not compile

Which of these prototypes is the best one to use in this circumstance?
 	
int main()
{
string str{"To be or not to be."};
cout << "Most common letter is "
<< mostCommon(str) << endl;
}
	char mostCommon(const string&);

Which of these prototypes is the best one to use in this circumstance?
 	
int main()
{
string str{"TO BE OR NOT TO BE"};
properCase(str);
cout << str << endl;
}
	void properCase(string&);

Examine this code. Which is the best prototype?
 	
string s = "dog";
cout << upper(s) << endl; // DOG
cout << s << endl; // dog
	string upper(const string&)

Examine this code. Which is the best prototype?
 	
string s = "dog";
upper(s);
cout << s << endl; // DOG
	string upper(const string&)

Arguments passed to a function that has a non-constant reference parameter must be:	lvalues

Arguments passed to a function that has a constant reference parameter must be:	either lvalues or rvalues are fine
The pattern of parameter types and order is called the function's:	signature

What prints here?
 	
int i = 5;
while (--i) cout << i;
cout << endl;
	4321

What prints here?
 	
int i = 5;
while (i--) cout << i;
cout << endl;
	43210

What prints here?
 	
int i = 5;
while (i) cout << --i;
cout << endl;
	43210

What prints here?
 	
int i = 5;
while (i) cout << i--;
cout << endl;
	54321

What prints here?
 	
int i = 5;
while (i); cout << i--;
cout << endl;
	Infinite loop

End a block of source code	@endcode
Meaning of value returned from a function	@return
Required to document functions, global variables and constants	@file
Begin a block of source code	@code
Your name	@author
Information about the library	@version
When was it created?	@date
Name and meaning for a parameter	@param
Which of these documentation tags are used in a file comment?	@version @author @date @file
Which of these documentation tags are used in a function comment?	@return @param @endcode @code
What kind of error is this? ex1.cpp:7:10: error: expected ';' after expression a = 4 ^ ;	Syntax error (mistake in grammar)
What kind of error is this? ex1.cpp:6:5: error: use of undeclared identifier 'a' a = 4; ^	Compiler error (something is missing when compiling)
What kind of error is this? ex1.cpp:6:12: error: no viable conversion from 'int' to 'string' string a = 15; ^ ~~	Type error (wrong initialization or assignment)
What kind of error is this? ex1.cpp:7:9: warning: missing terminating '"' character a = "hello world'; ^ ex1.cpp:7:9: error: expected expression	Syntax error (mistake in grammar)

string s = "12345";
int i = 1;
while (i < 5)
{
cout << s.substr (i, 1);
i++;
}	

What is the output of the following?
 	
string s = "abcde";
int i = 1;
while (i < 5)
{
cout << s.substr (i, 1);
i++;
}
	bcde

What is the output of the following?
 	
int i = 1;
while (i < 10)
{
cout << i << " ";
i = i + 2;
if (i == 5)
{
i = 9;
}
}
	1 3 9

What is the output of the following?
 	
int i = 1;
while (i <= 10)
{
cout << "Inside the while loop" << endl;
i = i * 11;
}
	"Inside the while loop" will be displayed only once.

What is the output of the following?
 	
int i = 1;
int sum = 0;
while (i <= 11)
{
sum = sum + i;
i++;
}
cout << "The value of sum is " << sum;
	The value of sum is 66

What is the output of the following?
 	
int i = 0;
while (i != 11)
{
cout << i << " ";
i = i + 2;
}
	0 2 4 6 8 10 12 14 .... (infinite loop)

What is the output of the following?
 	
int i = 1;
int sum = 0;
while (i <= 13)
{
sum = sum + i;
i = i + 3;
}
cout << "The value of sum is " << sum;
	The value of sum is 35

How many times will this display "So far so good"?
 	
int i = 0;
while (i != 15)
{
cout << "So far so good" << endl;
i++;
}
	15 times

int i = 1;
while (i < 20)
{
cout << i << " ";
i = i + 2;
if (i == 15)
{
i = 19;
}
}	

What is the output of the following?
 	
int i = 0, j = 0;
while (i < 125)
{
i = i + 2;
j++;
}
cout << j << endl;
	63

Header files must explicitly qualify each name from the standard library with std:: Header files may use the statement using namespace std;	True False
An undefined error message is a linker error. An undefined error message is a compiler error	True False
An undeclared error message is a run-time error An undeclared error message is a linker error	False False
Implementation files may use the statement using namespace std; Implementation files must explicitly qualify each name from the standard library with std::	True False
Parameter names are optional in the function prototype Parameter names are optional in the function definition	True False
A tool named Doxygen is often used to generate HTML user docs from C++ code. If a prototype in a header file has a parameter that is a library type, the header file must #include the appropriate library header.	True True

Which prototypes in the following header file contain errors?
 	
#ifndef EXAMPLE_H
#define EXAMPLE_H
#include <string>
 	
string f1(int a);
int f2(double);
void f3(std::string& s, int n);
double f4();
 	
#endif
	f1

Which prototypes in the following header file contain errors?
 	
#ifndef EXAMPLE_H
#define EXAMPLE_H
 	
string f1(int a);
int f2(double);
void f3(std::string& s, int n);
double f4();
 	
#endif
	f1
	f3

#ifndef EXAMPLE_H
#define EXAMPLE_H
#include <string>
 	
std::string f1(int a);
int f2(double);
void f3(std::string& s, int n);
double f4();
 	
#endif	

Which of these are dependencies? EXE=digit-tester OBJS=client.o digits.o $(EXE): $(OBJS) $(CXX) $(CXXFLAGS) $(OBJS) -o $(EXE)	digits.o client.o
Which of these are targets? EXE=digit-tester OBJS=client.o digits.o $(EXE): $(OBJS) $(CXX) $(CXXFLAGS) $(OBJS) -o $(EXE)	$(EXE) digit-tester

How many lines of output are printed?
 	
int i = 0;
while (i != 9)
{
cout << "Loop Execution" << endl;
i++;
}
	9

What is the output of the following?
 	
int i = 0;
while (i != 9)
{
cout << i << " ";
i = i + 2;
}
	0 2 4 6 8 10 12 14 .... (infinite loop)

What is the output of the following?
 	
int i = 1;
while (i != 9)
{
cout << i << " ";
i++;
if (i = 9)
{
cout << "End";
}
}
	1 End

How many lines of output are printed?
 	
int count = 0;
while (count != 9)
{
cout << "Monster Mash" << endl;
if ((count % 2) == 0)
{
count++;
}
else
{
count--;
}
}
	Infinite

What is the output of the following?
 	
bool token = false;
while (token)
{
cout << "Hello World!" << endl;
}
	No output

bool token1 = true;
while (token1)
{
for (int i = 0; i < 5; i++)
{
cout << "Hello there" << endl;
}
token1 = false;
}	

What is the output of the following?
 	
bool val1 = true;
bool val2 = false;
while (val1)
{
if (val1)
{
cout << "Hello" << endl;
}
val1 = val2;
}
	"Hello" will be displayed only once.

Which line in the function "skeleton" below contains an error?
 	
#include "digits.h" // 1.
int firstDigit(int n); // 2.
{ // 3.
return 0; // 4.
} // 5.
	// 2.

Which line in the function "skeleton" below contains an error?
 	
#include "digits.h" // 1.
int firstDigit(int n) // 2.
{ // 3.
return 0; // 4.
}
	None of these

Which line in the function "skeleton" below contains an error?
 	
#include "borgia.h" // 1.
void primoTiara(int n) // 2.
{ // 3.
return 0; // 4.
} // 5.
	// 4.

What kind of error is this? ex1.cpp:7: undefined reference to `f()'	Linker error (something is missing when linking)
What kind of error is this? ~/workspace/ $ ./ex1 The Patriots won the 2018 Super Bowl	None of these
What kind of error is this? terminate called after throwing an instance of 'std::out_of_range'	Runtime error (throws exception when running)
What kind of error is this? Segmentation fault	Operating system signal or trap
In a library, the implementation file:	consists of function definitions
In a library, the interface file:	consists of declarations or prototypes
In a library, the client or test program:	consists of function calls
In a library, the makefile:	consists of instructions that produce the executable
In a client file you should compare your function's value to the _____?	expected value
In a client file, the value returned from calling your function is the_____?	actual value
Loops that do some processing and then compare the results against a boundary condition are called _____?	limit loops
An incomplete, yet compilable, linkable and executable function is called a _____ ?	stub
Which of these program organization schemes does not work?	Call your functions and define them afterwards.
Which of these may go into a header file?	function prototypes constant definitions

When you call a function, the compiler must know:	the name of the function the type of each argument the kind of value returned if any
Header guards:	end with the directive #endif includes the directive #define go in every interface file start with the directive #ifndef
Executable Object file Library file Interface file Project file Client file Implementation file	digit-tester digits.o libdigits.a digits.h makefile digit tester.cpp digits.cpp
The input stream member function for reading a character at a time is named:	get()
Assume you have a char variable named ch. How do you read one character from input?	cin.get(ch);
The expression cin.get(ch) does which of these?	reads the next character in input and stores it in ch returns a reference to cin that can be tested
Assume you have a char variable named ch. How do you "unread" a character already read?	cin.putback(ch);
Assume you have a char variable named ch. How do you write one character to output?	cout.put(ch);
Complete the following code in the echo filter program. char ch; while (cin.get(ch)) _____;	cout.put(ch)

Complete the following code in the lower filter program.
 	
char ch;
while (cin.get(ch))
cout.put(_____);
	tolower(ch)

Complete the following code in the upper filter program.
 	
char ch;
while (cin.get(ch))
cout.put(_____);
	toupper(ch)

Complete the following code in the echo filter program.
 	
char ch;
while (_____)
cout.put(ch);
	cin.get(ch)

Assume the user types "brown cow" when this code runs. What type is ch2? char ch1; auto ch2 = cin.get(ch1);	istream&
Assume the user types "brown cow" when this code runs. What prints? int n; if (cin >> n) cout << "X\n"; else cout << "Y\n";	Y
Assume the user types "brown cow" when this code runs. What is stored in ch2? char ch1; auto ch2 = cin.get(ch1);	cin
Assume the user types "brown cow" when this code runs. What prints? char c; cout.put(cin.get(c));	Does not compile

char c; cout << cin.get(c) << endl;	
When using cin >> ch; to read a character, leading whitespace is skipped. When using cin >> ch; to read a character, leading whitespace is not skipped.	True False
Calling cout.put(65) prints the character 'A' on output Calling cout.put(65) prints the number 65 on output Calling cout.put(65) is illegal. Your code will not compile. Calling cout.put(65.35) is illegal. Your code will not compile	True False False False
When using the get() member function to read a character, leading whitespace is not skipped When using the get() member function to read a character, leading whitespace is skipped.	True False
A process filter does something to the characters it encounters A process filter learns something about the stream by examining characters	True False
The expression cin.get(ch) returns a reference to the input stream The expression cin.get(ch) returns the next character from input	True False
A state filter learns something about the stream by examining characters A state filter does something to the characters it encounters	True False
Counting the number of words in input by counting word transitions is an example of a state filter Counting the number of words in input by counting word transitions is an example of a process filter.	True False
You can test if an I/O operation succeeded by explicitly calling the stream's fail() member function To test if an I/O operation succeeded you must explicitly call the stream's fail() member function	True False
Calling cout.put(c) converts its argument, c, to a character.	True
Calling cout.put("A") is illegal. Your code will not compile.	True
When a stream is converted to a Boolean condition, its fail() member function is implicitly called	True
When using the get() member function, a stream will fail only if there are no characters left in the input stream.	True
Programs that process streams of characters are called text _____.	filters
Which of these are not process filters?	compress input by turning off echo when reading blank spaces print one sentence per line counting word transitions
Which of these are not state filters?	translating data from one form to another search for a particular value in a stream copy a file
Assume you have a char variable named ch. How do you look ahead before reading a character?	cin.peek();

Assume you have a char variable named ch. How do you look ahead before reading a character?
 	
2 QUESTIONS
	cin.get(ch);
	cin.unget(ch);
	cin.putback(ch);
	cin.seek(ch);
	cin.peek(ch);
	
	-- > None of these

Which line runs the prt program and stores its output in a new file named x.data?	./prt > x.data
Which line runs the dmm program and adds its output to a file named x.data?	./dmm >> x.data
Which line runs the dd program and sends its errors to file named z.data?	./dd 2> z.data
Which line runs a.out getting its input from in.txt and appending its output to the file out.txt?	./a.out > in.txt >> out.txt
Which line runs a.out getting its input from in.txt and sending its output to the new file out.txt?	./a.out > out.txt < in.txt
Append output to a file named z	X
Discard both output and errors	rm x > /dev/null/2>&1
Write output to a new file named z	X
Read the input from the file named z	cat < z
Write errors to a new file named z	cat x 2>z
Send the output to the input of the program named z	date I z
Which line runs the dom program and sends both output and errors to file named v.data?	./dom > v.data 2>&1
Has a single char& parameter	get()
Returns the last character read to the input stream	unget()
Examines, but does not read the next character in an input stream	peek()
Replaces the last character read with any character	putback()
Called implicitly when an input statement is used as a test condition.	fail()
A predicate function	isalpha()
Converts its value argument to a character and sends it to output.	put()
Which line runs a.out getting its input from in.txt and sending its output to the file out.txt, and its errors to the file err.txt?	./a.out < in.txt > out.txt 2> err.txt
Indefinite limit loop that reduces its input	while (n!=0) {n/=2;}
Indefinite limit loop that uses successive approximations	while(abs(g1-g2) >= EPSILON) {...}
Counter-controlled symmetric loop for producing a sequence of data	for (int i = 12; i <= 19; i ++) {...}
Indefinite data loop that uses raw input	while(cin.get(ch)) {...}
Counter-controlled asymmetric loop for processing characters	for (size_t i = 0, len = s.size(); i < len; i++) {...}
Iterator loop that may change its container	for(auto&e : col) {...}
Iterator loop that cannot change its container	for(auto e: col) {...}
Counter-controlled loop for processing substrings	for(size_t i=4, slen =4; len = s.size(); i <len; i++) {...}
Indefinite data loop that uses formatted input	while(cin >> n)
A loop that reads data until some special value is found is called a:	sentinel loop
Which of these is not a technique for implementing a sentinel loop?	the counter-controlled pattern
What Java and other OO languages call a subclass, C++ calls a _____.	derived class
Stream arguments to a function should:	be as general as possible (istream and ostream)
Stream arguments to a function should always be passed:	by reference

The file temp.txt contains "Orange Coast College". What prints?
 	
ifstream in("temp.txt");
char c;
while (in.get(c))
{
if (isupper(c))
cout << toupper(c);
}
	OCC

Which line opens the file in.txt for reading?	ifstream in("in.txt");
Which line opens the file input.txt for reading?	ifstream in("input.txt");
Create an input file stream object named in and open the text file "tuba.txt", using a single statement.	ifstream in("tuba.txt");
Create an output file stream object named out.	ofstream out;
Which line opens the file out.txt for writing?	ofstream out; out.open("out.txt");
Create an output file stream object named out and open the text file "expenses.dat", using a single statement.	ofstream out("expenses.dat");
Use the output stream object named out to create the text file on disk named "totals.txt".	out.open("totals.txt");
Establish an association between the input stream object named in, and the text file on disk named "pets.txt".	in.open("pets.txt");

Which line reads a single word from the istream named in into the string variable word?
 	
word = in.next();
in.get(word);
getline(in, word);
in << word;
None of these
	None of these

None of these	

The file temp.txt contains "If I saw an Aardvark, I would scream!". What prints?
 	
ifstream in("temp.txt");
char c;
int i = 0;
while (in.get(c))
{
if (tolower(c) == 'a') i++;
}
cout << i << endl;
	6

The return value of the getline() function is an input stream object The return value of the getline() function is a string object.	True False
When writing a function with stream parameters, always use the most general type of stream that meets the specification When writing a function with stream parameters, always use the most specific type of stream that meets the specification	True False
The cout object is an instance of the ostream class. The cout object is an instance of the ofstream class	True False
A loop that reads data until the input stream signals that it is done is called a data loop A loop that reads data until the input stream signals that it is done is called a sentinel loop	True False
In the primed loop pattern, you read data before the loop and at the end of the loop. In the primed loop pattern, you use Boolean flag to signal when the sentinel is found In the primed loop pattern, you use a break statement to exit the loop when the sentinel is found	True False False
The getline() function is a non-member function in the string library The getline() function is a member function in the string class The getline() function is a member function in the istream class.	True False False

To use a disk file as a data stream source or sink, use the <fstream> header	
To use a disk file as a data stream source or sink, use the <ifstream> header	False
To use a disk file as a data stream source or sink, use the <ofstream> header	False
Unformatted I/O means that you read and write data character-by-character	True
Unformatted I/O means that you read and write data line-by-line	False
Formatted I/O means that you read and write data token-by-token	True
Formatted I/O means that you read and write data line-by-line	False
The C++ term for what is called a superclass in other languages is base class	True
The C++ term for what is called a superclass in other languages is derived class	False
The cin object is an instance of the istream class	True
The cin object is an instance of the ifstream class	False
Stream parameters should always be passed to functions by reference	True
Stream parameters should always be passed to functions by const reference	False
In the flag-controlled-pattern, you use Boolean variable to signal when the sentinel is found	True
In the flag-controlled-pattern, you use a break statement to exit the loop when the sentinel is found.	False
In the flag-controlled-pattern, you read data before the loop and at the end of the loop	False
In the loop-and-a-half, you use a break statement to exit the loop when the sentinel is found	True
In the loop-and-a-half, you use Boolean variable to signal when the sentinel is found	False
In the loop-and-a-half pattern, you read data before the loop and at the end of the loop.	False
If an input stream's file is missing when you try to open it, its fail() member function returns true	True
If an input stream's file is missing when you try to open it, its fail() member function returns false	False
If an output stream's file is missing when you try to open it, its fail() member function returns false.	True
To use strings as a data stream source or sink, use the <sstream> header	True
To use strings as a data stream source or sink, use the <stringstream> header	False
The C++ term for what is called a subclass in other languages is derived class	True
The C++ term for what is called a subclass in other languages is base class	False
A loop that reads data until some special value is found is called a sentinel loop.	True
A loop that reads data until some special value is found is called a data loop.	False
To read a line of text, you include the header file <string>	True
A token is a "chunk of meaningful data".	True
In the C++ stream hierarchy, the base class of the ifstream class is:	istream

In the C++ stream hierarchy, the base class of the ostream class is:	ios
In the C++ stream hierarchy, base class of the istream class is:	ios
In the C++ stream hierarchy, the base class of the stringstream class is:	iostream
In the C++ stream hierarchy, the base class of the fstream class is:	iostream
Read and write characters to memory using streams Connect a disk file to an input or output stream Use the predefined stream objects cin and cout Determine the category of a character Modify the way that memory is converted to characters on input or output	sstream fstream iostream cctype iomanip

Which fragment completes this code segment?
 	
string fmt(double n, int decimals)
{
ostringstream out;
out << fixed << setprecision(decimals);
out << n;
return _____;
}
}
	out.str()

After writing data to an ostringstream object named os, you can retrieve the string it contains by using:	os.str()

What does this code do?
 	
ifstream in("temp.txt");
char x;
int i{0};
while (in.get(x)) i++;
cout << i << endl;
	Counts the number of characters in the file

What does this code do?
 	
ifstream in("temp.txt");
string x;
int i{0};
while (getline(in, x)) i++;
cout << i << endl;
	Counts the number of lines in the file

What does this code do?
 	
ifstream in("temp.txt");
string x;
int i{0};
while (in >> x) i++;
cout << i << endl;
	Counts the number of words in the file

Which of the following loop patterns are used here?
 	
size_t pos = 0;
char ch;
in.get(ch);
while (ch != 'Q')
{
pos++;
in.get(ch);
}
	primed loop
	sentinel loop

Which of the following loop patterns are used here?
 	
int upper = 0;
char ch;
while (in.get(ch))
{
if (ch >= 'A' && ch <= 'Z')
upper++;
}
	inline test
	data loop

Which of the following loop patterns are used here?
 	
int n;
in >> n;
while (abs(n))
{
out << n % 4 << endl;
n /= 4;
}
	limit loop

auto len = str.size(); while (len) out << str.at(--len);	

Which of the following loop patterns are used here?
 	
string s{"hello CS 150"};
for (auto e : s)
{
if (toupper(e))
out.put('x');
}
	iterator or range loop

Which of the following loop patterns are used here?
 	
string s{"hello CS 150"};
for (auto e : s)
{
if (toupper(e)) break;
}
	iterator or range loop
	
	loop-and-a-half

Which of the following loop patterns are used here?
 	
string s{"Hello CS 150"};
while (s.size())
{
if (s.at(0) == 'C') break;
s = s.substr(1);
}
cout << s << endl;
	counter-controlled loop
	
	loop-and-a-half
	
	sentinel loop

After opening the input stream in, which of these cannot be used to see if the file was successfully opened?	if (in.opened()) {/ **opened ok** /}

This loop:
 	
char c;
while (in.get(c))
{
cout << c << endl;
}
	illustrates raw character I/O

This loop:
 	
char c;
while (c = in.get())
{
cout << c << endl;
}
	illustrates line-based stream processing

This loop:
 	
string str;
while (getline(in, str))
{
cout << str << endl;
}
	illustrates line-based stream processing

This loop:
 	
string str;
while (in >> str)
{
cout << str << endl;
}
	illustrates token-based stream processing

The file grades.txt contains lines of text that look like this: Smith 94 Jones 75 . . . Each line of text contains the student's name (a single word) and an integer score. What is the legal way of reading one student's information, given the following code? string name; int score; ifstream in("grades.txt");	in >> name >> score;

ifstream in("expenses.txt");
char c;
while (in.get(c))
{
if (isdigit(c)) {
in.unget();
double n;
in >> n;
cout << n << 'x';
}
}	

The file expenses.txt contains the line: Hotel, 3 nights. $ 1,750.25. What prints?
 	
ifstream in("expenses.txt");
char c;
while (in.get(c))
{
if (isdigit(c)) {
in.unget();
int n;
in >> n;
cout << n << 'x';
}
}
	3x1x750x25x

Assume that the file scores.txt does not exist. What happens? ofstream out("scores.txt"); out << "Peter" << " " << 20 << endl; out << "John" << " " << 50 << endl;	Creates a new file, scores.txt and writes two lines of text.
Which line represents the necessary bounds in this loop? 1. string s("Hello CS 150"); 2. while (s.size()) 3. { 4. if (s.at(0) == 'C') break; 5. s = s.substr(1); 6. } 7. cout << s << endl;	2
Which line represents the intentional bounds in this loop? 1. string s("Hello CS 150"); 2. while (s.size()) 3. { 4. if (s.at(0) == 'C') break; 5. s = s.substr(1); 6. } 7. cout << s << endl;	4
Which line advances the loop? 1. string s("Hello CS 150"); 2. while (s.size()) 3. { 4. if (s.at(0) == 'C') break; 5. s = s.substr(1); 6. } 7. cout << s << endl;	5
What header file to you need to include to use the standard C++ error-handling classes?	<stdexcept>
The logic_error and runtime_error classes are defined in the header file ___.	stdexcept

What prints?
 	
string s("hello");
try {
auto x = s.at(s.size()); ☼
cout << "one" << endl;
}
catch (const string& e) { cout << "two\n"; }
catch (exception& e) { cout << "three\n"; }
catch (...) { cout << "four\n"; }
	three

```
string s("hello");
try {
if (s.size() > 20) throw 42;
if (isupper(s.back())) throw "goodbye";
if (s == "Hello") throw string("hello");
s.at[s.size()] = 'x'; 🔅
cout << "one\n";
}
catch (const int& e) { cout << "two\n"; }
catch (const string& e) { cout << "three\n"; }
catch (exception& e) { cout << "four\n"; }
catch (...) { cout << "five\n"; }
```

What prints?

```
string s("hello");
try {
if (s.size() > 2) throw s.size(); 🔅
if (islower(s.back())) throw s.back(); 🔅
if (s == "hello") throw string("hello");
s.at(s.size()) = 'x';
cout << "one\n";
}
catch (const int& e) { cout << "two\n"; }
catch (const string& e) { cout << "three\n"; }
catch (exception& e) { cout << "four\n"; }
catch (...) { cout << "five\n"; }
```

➢ I F (s.size() > 2) && throw s.size() && throw s.back()

five

What prints?

```
string s("hello");
try {
if (s.size() > 5) throw s.size(); 🔅
if (isupper(s.back())) throw s.back(); 🔅
if (s == "hello") throw string("hello");
s.at(s.size()) = 'x';
cout << "one\n";
}
catch (const string& e) { cout << "two\n"; }
catch (exception& e) { cout << "three\n"; }
catch (...) { cout << "four\n"; }
```

➢ I F (s.size() > 5) && throw s.size() && throw s.back()

two

What prints?

```
string s("hello");
try {
if (s.size() > 2) throw 42; 🔅
if (islower(s.back())) throw "goodbye"; 🔅
if (s == "hello") throw string("hello");
s.at(s.size()) = 'x';
cout << "one\n";
}
catch (const int& e) { cout << "two\n"; }
catch (const string& e) { cout << "three\n"; }
catch (exception& e) { cout << "four\n"; }
catch (...) { cout << "five\n"; }
```

➢ I F (s.size() > 2) && throw 42; && throw "goodbye";

two

What prints?

```
string s("hello");
try {
if (s.size() > 20) throw 42; 🔅
if (islower(s.back())) throw "goodbye"; 🔅
if (s == "hello") throw string("hello");
s.at(s.size()) = 'x';
cout << "one\n";
}
catch (const int& e) { cout << "two\n"; }
catch (const string& e) { cout << "three\n"; }
catch (exception& e) { cout << "four\n"; }
catch (...) { cout << "five\n"; }
```

➢ I F (s.size() > 20) && throw 42; && (islower(s.back())) throw "goodbye";

five

```
string s("hello");
try {
if (s.size() > 20) throw 42;
if (isupper(s.back())) throw "goodbye";
if (s == "Hello") throw string("hello");
s.at(s.size()) = 'x';
cout << "one\n";
}
catch (const int& e) { cout << "two\n"; }
catch (const string& e) { cout << "three\n"; }
catch (exception& e) { cout << "four\n"; }
catch (...) { cout << "five\n"; }
```

➢ I F (s.size() > 2) && throw 42; && (isupper(s.back())) throw "goodbye";

What is correct for # 1?

```
int main()
{
//1
{
string s = "hello";
cout << s.at(5) << endl;
}

// 2
// 3
( e)
{
cout << e. () << endl;
// 4
}

}
```

try

What is correct for # 2?

```
int main()
{
//1
{
string s = "hello";
cout << s.at(5) << endl;
}

// 2
// 3
( e)
{
cout << e. () << endl;
// 4
}

}
```

catch

What is correct for # 3?

```
int main()
{
//1
{
string s = "hello";
cout << s.at(5) << endl;
}

// 2
// 3
( e)
{
cout << e. () << endl;
// 4
}

}
```

exception&

{
//1
{
string s = "hello";
cout << s.at(5) << endl;
}
 	
// 2
// 3
(e)
{
cout << e. () << endl;
// 4
}
 	
}	

The C++11 standard library provides the function stoi() to convert a string to an integer. Which library is it found in?	string
What preprocessor directive is not used when you wish to create blocks of code that are only compiled under certain circumstances?	#define #ifdef #ifndef #if --> All of these may be used
Code that may cause an error should be placed in a _____ block and code that handles the error should be inside a _____ block?	try, catch
The class ___ is the base of the classes designed to handle exceptions	exception
A(n) ___ is an occurrence of an undesirable situation that can be detected during program execution	exception
What statement is used to signal other parts for your program that a particular error has occurred?	throw
The class ___ is designed to deal with illegal arguments used in a function call.	invalid_argument
What is the purpose of the throw statement?	It is used to pass control to an error handler when an error situation is detected.
The try block is followed by one or more ___ blocks.	catch
Which of the following blocks is designed to catch any type of exception?	catch(...){ }
The function ___ returns a string containing an appropriate message.	what
A catch block can have, at most, ___ catch block parameter(s).	one
What happens when this code fragment runs in C++ 11? cout << sqrt(-2) << endl;	sqrt() returns a not-a-number error value
Variables tested with the #if preprocessor directive are created using #define	True
Without try and catch, the throw statement terminates the running program	True
A try block is a block of code where runtime or logical errors may occur	True
A catch(...) will catch any kind of thrown exception	True
Functions with generic parameters are known as function templates.	True
A completion code is a special return value that means "the function failed to execute correctly."	True
Calling a function like to_string(3.5) is known as implicit instantiation	True
To use different versions of a function depending on the platform is called conditional compilation.	True
Building your code with more than one copy of a function leads to a clash of symbols.	True
A template function may be defined in a header file.	True
The predefined constant _cpluplus indicates which version of the C++ standard is being used	True
One of the main problems with the completion code strategy of error handling is that callers can ignore the return value without encountering any warnings	True
Calling a function like to_string<int>(3.5) is known as implicit instantiation.	False

The preprocessor operates on code after it has been compiled.	False
The directives #if defined(symbol) and #ifdef symbol mean, essentially, the same thing	True
The directives #if defined(symbol) and #ifndef symbol mean, essentially, the same thing.	False
A catch block may handle exception classes, as well as errors where int or string are thrown	True
A catch block may only handle objects from classes derived from exception or logic_error	False
A catch block specifies the type of exception it can catch and immediately terminates the program	False
A catch block is a block of code where runtime or logical errors may occur	False
You can report a logical error encountered in your code by using the throw keyword	True
You can report a syntax error encountered in your code by using the throw keyword	False
Functions with generic parameters may use the keyword class or the keyword typename for their type parameters	True
Functions with generic parameters may use the keyword class or the keyword struct for their type parameters	False
The #if preprocessor directive can compare integers	True
The #if preprocessor directive may compare double literals but not variables	False
The standard library version of sqrt(-2) returns the not-a-number error code	True
The standard library version of sqrt(-2) throws a runtime exception because there is no possible answer	False
You compiler or contains constants that can be used to identify the platform you are compiling on	True
A specialized error handling block of code, is called a catch block	True
A specialized error handling block of code, is called a try block	False
The standard library version of stoi("UB-40") throws a runtime exception because there is no viable conversion	True
The standard library version of stoi("UB-40") returns the not-a-number error code.	False
The order of the catch blocks does not affect the program.	False
If no exception is thrown in a try block, all catch blocks associated with that try block are ignored.	True
When you throw an exception, control immediately jumps out of the current try block.	True
The preprocessor operates on code before it has been compiled.	False
The statement #if abs(-3) > 2 is legal.	False
A template function may be declared in a header file but must be defined in an implementation file.	False
The heading of a try block can contain ellipses in place of a parameter	False
When you throw an exception, control immediately returns from the current function	False
The line: ifstream in("x"); throws a runtime exception if a file x cannot be found	False
What happens when this code fragment runs? cout << stoi("12") << endl;	stoi() returns 12

cout << stoi("one") << endl;	
Which of the following statements throws a valid exception in C++?	throw 2;
Suppose you have written a program that inputs data from a file. If the input file does not exist when the program executes, then you should choose which option?	Terminate the program.
What happens when this code fragment runs? istringstream in("12.5"); int n; in >> n;	n is set to 12
What happens when this code fragment runs? istringstream in("12"); int n; in >> n;	n is set to 12
What happens when this code fragment runs? istringstream in(".5"); int n; in >> n;	It sets an error state in in.
What happens when this code fragment runs in C++ 11? istringstream in("one"); int n; in >> n;	It sets an error state in in.
To deal with logical errors in a program, such as string subscript out of range or an invalid argument to a function call, several classes are derived from the class ___.	logic_error

Which line fails to work correctly?
 	
template <typename T>
void print(const T& item)
{
cout << item << endl;
}
	ANSWER --> None of these
	print(2 + 2);
	print(string("goodbye"));
	print(3 + 2.2);
	print("hello");

Assume s1 and s2 are C++ string objects. Which of these calls is illegal?
 	
template <typename T>
void addem(T a, T b)
{
cout << a << " + " << b << "->"
<< (a + b) << endl;
}
	addem(1.5, 2);

Which call below produces 5?
 	
template <typename T>
void addem(T a, T b)
{
cout << a << " + " << b << "->"
<< (a + b) << endl;
}
	addem<int>(3, 2.5);

Assume s1 and s2 are C++ string objects. Which of these calls is illegal?
 	
template <typename T>
void addem(T a, U b)
{
cout << a << " + " << b << "->"
<< (a + b) << endl;
}
	ANSWER --> None of these
	addem(1.5, 2);
	addem(s1, s2);
	addem(3, 4)
	addem(4.5, 5.5);

What happens when this code fragment compiles and runs? #define N #ifdef N cout << "Hello"; #else cout << "Goodbye"; #endif	prints "Hello"
What happens when this code fragment compiles and runs? #define N #ifndef N cout << "Hello"; #else cout << "Goodbye"; #endif	prints "Goodbye"

#if __APPLE__
istringstream in(" .75");
int n = 3;
in >> n;
#endif	

Complete the code fragment below, which is designed to throw an illegal_length exception if string variable accountNumber has more than seven characters.
 	
if (accountNumber.size() > 7)
{
_____;
}
	throw illegal_length("Account number exceeds maximum length");

The following definition: vector<double> data;	creates a vector of size 0
The following definition: vector<double> v{3, 5};	creates a vector of [3.0, 5.0]
The following definition: vector<double> v(3, 5);	creates a vector of [5.0, 5.0, 5.0]
What prints? vector<int> v{1, 2, 3, 4, 5}; cout << v.pop_back() << endl;	Nothing; compile-time error

What prints?
 	
vector<int> v{1, 2, 3, 4, 5};
v.pop_back();
cout << v.front() << endl;
	1

What prints?
 	
vector<int> v{1, 2, 3, 4, 5};
v.pop_back();
cout << v.back() << endl;
	4

What prints?
 	
void f(vector<int> v)
{
v.at(0) = 42;
}
int main()
{
vector<int> x{1, 2, 3};
f(x);
cout << x.at(0) << endl;
}
	1

What prints?
void f(vector<int>& v)
{
v.at(0) = 42;
}
int main()
{
vector<int> x{1, 2, 3};
f(x);
cout << x.at(0) << endl;
}
	42

What prints?
 	
void f(const vector<int>& v)
{
v.at(0) = 42;
}
int main()
{
vector<int> x{1, 2, 3};
f(x);
cout << x.at(0) << endl;
}
	Nothing; compile-time error.

What does this code do?
 	
int x = 0;
vector<int> v{1, 3, 2};
for (auto e : v) e += x;
cout << x << endl;
	prints 0

int x = 0;
vector<int> v{1, 3, 2};
for (auto e : v) x = e;
cout << x << endl;	

What does this code do?
 	
int x = 0;
vector<int> v{1, 3, 2};
for (auto e : v) x += e;
cout << x << endl;
	Sums the elements in v
	Prints 6

What is stored in data after this runs? vector<int> data{1, 2, 3}; data.pop_back();	None of these
What is the size of data, after this runs? vector<int> data; data.push_back(3);	1
What is stored in data after this runs? vector<int> data{1, 2, 3}; data.erase(v.begin());	[2, 3]
What is stored in data after this runs? vector<int> data{1, 2, 3}; data.front();	[1, 2, 3]
What is stored in data after this runs? vector<int> data{1, 2, 3}; data.back();	[1, 2, 3]
What is stored in data after this runs? vector<int> data{1, 2, 3}; data.clear();	[]
What is stored in data after this runs? vector<int> data{1, 2, 3}; data.push_back(0);	[1, 2, 3, 0]
What is stored in data after this runs? vector<int> data{1, 2, 3}; data.pop_back(0);	None of these

Which of these are true?
 	
int main()
{
vector<int> v{1, 2, 3};
for (const auto& e : v) e = 0;
cout << v.at(0) << endl;
}
	Code will not compile

Which of these are true?
 	
int main()
{
vector<int> v{1, 2, 3};
for (auto i = v.size() - 1; i >= 0; i--) // out of range for >=
cout << v.at(i) << " ";
cout << endl;
}
	Crashes when run
	
	Prints 3 2 1
	
	Issues a compiler warning, but no error

Which of these are true?
 	
int main()
{
vector<int> v{1, 2, 3};
for (auto& e : v) e = 0;
cout << v.at(0) << endl;
}
	Prints 0

Which of these are true?
 	
int main()
{
vector<int> v{1, 2, 3};
for (auto e : v) e = 0;
cout << v.at(0) << endl;
}
	Prints 1
	
	Code runs but has no effect on v

int main()
{
vector<int> v{1, 2, 3};
for (auto i = v.size() - 1; i >= 0; i--)
cout << v[i] << " ";
cout << endl;
}
	Issues a compiler warning, but no error
	
	Prints 3 2 1

Which of these are true?
 	
int main()
{
vector<int> v{1, 2, 3};
for (auto i = v.size(); i > 0; i--)
cout << v.at(i) << " ";
cout << endl;
}
	crashes when runs

Which line of code can be added to print the value 4?
 	
int main()
{
struct S {int a, b; };
vector<S> v;
S s{3, 4};
v.push_back(s);
// Add code here
}
	cout << v.at(0).b << endl;

Assume vector<double> speed(5); Which line throws a run-time error?	ANSWER --> None of these
	
	cout << speed[speed.size()];
	speed[0] = speed.back()
	speed.front() = 12;
	speed.erase(speed.begin());

Which defines a vector to store the salaries of ten employees?	vector<double> salaries(10);
The following code is logically correct. What is the semantically correct prototype for mystery()? vector<double> v; mystery(v);	void mystery(vector<int>&);
The following code is logically correct. What is the semantically correct prototype for mystery()? vector<double> v{1, 2, 3}; mystery(v);	Either mystery(const vector<int>&); or mystery(vector<int>&); could be correct.

Which line will not compile?
 	
int main()
{
vector<int> v{1, 2, 3};
auto size = v.size();
 	
cout << v.back() << endl; // 1.
cout << v.front() << endl; // 2.
cout << v.at(0) << endl; // 3.
cout << v.at(size) << endl; // 4.
cout << v.pop_back() << endl; // 5.
}
	5

Which line prints 3?
 	
int main()
{
vector<int> v{1, 2, 3};
auto size = v.size();
 	
cout << v.back() << endl; // 1.
cout << v.front() << endl; // 2.
cout << v.at(0) << endl; // 3.
cout << v.at(size) << endl; // 4.
cout << v.pop_back() << endl; // 5.
}
	1

Which statement is false? The elements in a vector:	ANSWER → None of these Are accessed by using an index or subscript Each use the same amount of memory Are are all of the same type Are homogeneous

int main()
{
vector<int> v{1, 2, 3};
auto size = v.size();
 	
cout << v.back() << endl; // 1.
cout << v.front() << endl; // 2.
cout << v.at(0) << endl; // 3.
cout << v.at(size) << endl; // 4.
cout << v.pop_back() << endl; // 5.
}	

Which lines have an identical effect?
 	
int main()
{
vector<int> v{1, 2, 3};
auto size = v.size();
 	
cout << v.back() << endl; // 1.
cout << v.front() << endl; // 2.
cout << v.at(0) << endl; // 3.
cout << v.at(size) << endl; // 4.
cout << v.pop_back() << endl; // 5.
}
	2 and 3

In C++ the parameterized collection classes are called _____?	templates
Classes that contain objects as elements are called?	collections

Assume vector<double> speed(5); Which line throws a runtime error?	None of these
	speed.erase(speed.begin());
	speed.front() = 12;
	speed[0] = speed.back()
	
	ANSWER → cout << speed.at(speed.size());

vector<int> v;	Creates the empty vector []
vector<int> v(1);	Creates the vector [0]
v.begin()	Points to the first element in v
v.back();	Returns a reference to the last element in v
v.erase(v.begin());	Removes the first element in v and shifts the rest to the left
v.pop_back()	Removes the last element in v
v[3];	Returns a reference to the fourth element in v with no range checking
vector<int>v(2,3);	Creates the vector [3,3]
vector<int>v[2, 3];	Creates the vector [2, 3]
v.push_back(3);	Adds a new element to the end of v
v.at(3);	Safely returns a reference to the fourth element in v

Assume vector<int> v; Writing cout << v.front(); throws a runtime exception. (false)

Assume the vector v contains [1, 2, 3]. v.erase(v.begin() + 2); changes v to [1, 2].

The declaration: vector<string> v(5, "bob"); creates a vector containing five string objects, each containing "bob".

In the declaration: vector<int> v; the word int represents the object's base type.

The elements of a vector are allocated contiguously.

vector subscripts begin at 0 and go up to the vector size - 1

The clear() member function removes all the elements from a vector.

The statement v.insert(v.end() + 1, 3) is undefined because end() + 1 points past the last element in the vector.

The statement v.insert(v.end(), 3) appends the element 3 to the end of the vector v.

Contiguous allocation means that the elements are stored next to each other in memory.

The push_back member function adds elements to the end of a vector.

Assume the vector v contains [1, 2, 3]. v.erase(v.begin()); changes v to [2, 3].

The declaration: vector<int> v(10); creates a vector object containing ten elements initialized to 0.

Assume the vector v contains [1, 2, 3]. v.pop_back(); changes v to [1, 2].

The term for classes with a base-type specification are parameterized classes.

Assume that v contains [1, 2, 3]. The result of writing cout << v[4]; is undefined.

The C++ term for classes like vector are template classes.

A vector subscript represents the element's offset from the beginning of the vector.

The declaration: vector<string> v{"bill", "bob", "sally"}; creates a vector containing three string objects.

The declaration: vector<int> v(10, 5); creates a vector object containing ten integers.

Assuming that Star is a structure, the declaration: vector<Star> stars(3); creates three default initialized Star objects.

The declaration: vector<string> v(5); creates a vector containing five empty string objects.

Assume the vector v contains [1, 2, 3]. v.erase(0); is a syntax error.

The declaration: vector<int> v; creates a vector object with no elements.

A vector represents a linear homogeneous collection of data.

Assume vector<double> v; Writing cout << v.back(); is undefined.

Elements in a vector are accessed using a subscript.

Assume that v contains [1, 2, 3]. The result of writing cout << v.at(4); throws a runtime exception.

The statement v.insert(v.begin(), 3) inserts the element 3 into the vector v, shifting the existing elements to the right.

Vector subscripts begin at 1 and go up to the vector size. The statement v.insert(v.end(), 3) is undefined because end() points past the last element in the vector. Assume that v contains [1, 2, 3]. The result of writing cout << v.at(4); is undefined. The C++ term for classes like vector are generic classes. The statement v.insert(v.begin(), 3) inserts the element 3 into the vector v, overwriting the exiting element at index 0. The push_back member function adds elements to the end of a vector as long as there is room for the elements. The declaration: vector<int> v(10); creates a vector object containing uninitialized elements. The declaration: vector<int> v(10, 5); creates a vector object containing five integers. The declaration: vector<string> v(5); creates a vector containing five null pointers. In the declaration: vector<int> v; the word vector represents the object's base type. The declaration: vector<int> v; creates a vector variable but no vector object. Assume that v contains [1, 2, 3]. The result of writing cout << v.at(4); is a compiler error. Vector subscripts begin at 1 and go up to the vector size. A vector consists of named members. The declaration: vector<int> v(10, 5); is illegal. Assume vector<double> v; Writing cout << v.back(); throws a runtime exception. Assume that v contains [1, 2, 3]. The result of writing cout << v[4]; is a compiler error. The declaration: vector<int> v = new vector<>(); creates a vector object with no elements. The pop_back member function adds elements to the end of a vector.	
Examine the following code (which is legal). What is the correct prototype for an aggregate output operator? struct Time { int hours{0}, minutes{0}, seconds{0}; };	ostream& operator<<(ostream& out, const Time& m);
Examine the following code (which is legal). What is the correct prototype for an aggregate output operator? struct Money { int dollars{0}, cents{0}; } m1, m2;	ostream& operator<<(ostream& out, const Money& m);
Examine the following code (which is legal). Which statement is illegal? struct Money { int dollars{0}, cents{0}; } m1, m2;	cout << m1 << endl;
Examine the following code (which is legal). Which statement is legal? struct Money { int dollars{0}, cents{0}; } m1, m2;	m1 = m2;
Examine the following code (which is legal). Which statement is correct? struct Rectangle { int length, width; };	Rectangle r;
The following is legal. Which is the correct way to access a data member in the Rectangle variable named r? struct Rectangle { int length, width; };	r.length
The structure and variable definitions are fine. Which statements are legal? struct Rectangle { int length, width; } big, small;	if (big.length == small.width) . . .
The following is legal. Which changes the length data member inside the variable big? struct Rectangle { int length, width; } big, little;	big.length = 10;

Examine the following code (which is legal). What changes are necessary to allow the statement if (m1 == m2) ... to compile?
 	
struct Money { int dollars{0}, cents{0}; } m1, m2;
 	
bool equals(const Money& lhs, const Money& rhs)
{
return lhs.cents == rhs.cents &&
lhs.dollars == rhs.dollars;
	The name of equals() must be changed to operator==

struct Money { int dollars{0}, cents{0}; } m1, m2;
 	
bool equals(const Money& lhs, const Money& rhs)
{
return lhs.cents == rhs.cents &&
lhs.dollars == rhs.dollars;
}	

Examine the following definition. What is the syntax error?
 	
struct Employee
{
long empID;
std::string lastName;
double salary;
}
	missing a semicolon after the structure definition

Examine the following definition. empID is a _____.
 	
struct Employee
{
long empID;
std::string lastName;
double salary;
};
	data member

Examine the following definition. Employee is the _____.
 	
struct Employee
{
long empID;
std::string lastName;
double salary;
};
	structure tag

Given the following structure and variable definitions, which data members are uninitialized?
 	
struct Employee
{
long empID{0};
std::string lastName;
double salary{0};
int age = 0;
};
 	
Employee bob;
	None of them (compiles)

Given the following structure and variable definitions, which data members are uninitialized?
 	
struct Employee
{
long empID;
std::string lastName;
double salary;
int age;
};
 	
Employee bob;
	salary
	age
	empID

Given the following structure and variable definitions, which data members are initialized?
 	
struct Employee
{
long empID;
std::string lastName;
double salary;
int age;
};
 	
Employee bob;
	lastName

Given the following structure and variable definitions, which data members are initialized?
 	
struct Employee
{
long empID;
std::string lastName;
double salary;
int age;
};
 	
Employee bob{};
	salary
	age
	lastName
	empID

struct Employee
{
long empID;
std::string lastName;
double salary;
int age;
};
 	
Employee bob{777, "Zimmerman"};	

Given the following structure and variable definitions, which data members are default initialized?
 	
struct Employee
{
long empID;
std::string lastName;
double salary;
int age;
};
 	
Employee bob{777, "Zimmerman", 5000000.0, 76};
	None of these

Given the following structure and variable definitions which statements are legal?
 	
struct Money
{
int dollars{0};
int cents{1};
};
Money payment;
	cout << payment.dollars;
	payment.cents = 5;

Given the following structure and variable definitions which statements are illegal?
 	
struct Money
{
int dollars{0};
int cents{1};
};
Money payment;
	payment{1} = 5;
	cout << Money.dollars;
	Money{1} = Money{0};

The structure and variable definitions are fine. Which statements are legal? struct R { int a, b; } a, b; struct Q { int a, b; } c, d;	c = d;
YOUDONOTNEEDTOREVIEWFORTRUE/FALSE	YOUDONOTNEEDTOREVIEWFORTRUE/FALSE

Structures are heterogeneous data types.

The built-in primitive data types such as int, char and double are scalar data types.

User-defined scalar types are created with the enum class keywords in C++.

User-defined types that contain a single value are called scalar types.

The standard library types such as string and vector are structured data types.

You may create a structure variable as part of a structure definition.

The following is an anonymous structure.
struct {int hours, seconds; } MIDNIGHT{0, 0};

Structure variables should be passed to functions by reference.

When passing a structure variable to a function, use non-const reference if the intent is to modify the actual argument.

The following code is legal.
struct {int hours, seconds; } MIDNIGHT{0, 0};

User-defined types that combine multiple values into a single type are called structured types.

A structure member may be a variable of a different structure type.

In C++, objects have value semantics; object variables contain the data members.

Structures data members may each have a different type.

C++ has two ways to represent records, the class and the struct.

This is the correct syntax for a C++ scoped enumeration.
enum class WEEKEND {SATURDAY, SUNDAY};

It is illegal to include the same struct definition multiple times, even if the definitions are exactly the same.

When passing a structure variable to a function, use const reference if the function should not modify the actual argument.

In Computer Science, a collection of variables that have distinct names and types is called a record.

This is the correct syntax for a C++ plain enumeration.
enum WEEKEND {SATURDAY, SUNDAY};

User-defined types that combine multiple values into a single type are called scalar types

It is legal to include the same struct definition multiple times, as long as the definitions are exactly the same.

In C++, objects have reference semantics; object variables refer to, but do not contain the data members.

A structure definition creates a new variable.

In C++, a collection of variables that have distinct names and types is called a record.

In C++, a collection of variables that have distinct names and types is called a structure.

User-defined types that contain a single value are called structured types.

This is the correct syntax for a C++ scoped enumeration.
enum WEEKEND {SATURDAY, SUNDAY};

Structure variables should be passed to functions by value.

User-defined scalar types are created with the struct or class keywords in C++.

Structures are homogenous data types.

User-defined types that combine multiple values into a single type are called scalar types.

Structures data members must all be of the same type.

When passing a structure variable to a function, use non-const reference if the function should not modify the actual argument.

The built-in primitive data types such as int, char and double are structured data types.

When passing a structure variable to a function, use const reference if the intent is to modify the actual argument.

The standard library types such as string and vector are scalar data types.

The following code is illegal.
struct {int hours, seconds; } MIDNIGHT{0, 0};

[1301] Which line below points ppi to pi?

```
int main()
{
double pi = 3.14159;
double *ppi;
// code goes here
// code goes here
}
```

ppi = π

[1302] Assume that ppi correctly points to pi. Which line prints the value stored inside pi?

```
int main()
{
double pi = 3.14159;
double *ppi;
// code goes here
// code goes here
}
```

cout << π
cout << ppi;
cout << &ppi;
cout << *pi;

→ None of these

[1303] Assume that ppi correctly points to pi. Which line prints the value stored inside pi?

```
int main()
{
double pi = 3.14159;
double *ppi;
// code goes here
// code goes here
}
```

cout << *ppi;

[1304] Assume that ppi correctly points to pi. Which line prints the address of ppi?

```
int main()
{
double pi = 3.14159;
double *ppi;
// code goes here
// code goes here
}
```

cout << &ppi;

```
int main()
{
double pi = 3.14159;
double *ppi;
// code goes here
// code goes here
}
```

[1306] The value for the variable a is stored:

```
int a = 1;
void f(int b)
{
int c = 3;
static int d = 4;
}
```

in the static storage area

[1307] The value for the variable b is stored:

```
int a = 1;
void f(int b)
{
int c = 3;
static int d = 4;
}
```

on the stack

[1308] The value for the variable c is stored:

```
int a = 1;
void f(int b)
{
int c = 3;
static int d = 4;
}
```

on the stack

[1309] The value for the variable d is stored:

```
int a = 1;
void f(int b)
{
int c = 3;
static int d = 4;
}
```

in the static storage area

[1310] The variable buf is a pointer to a region of memory storing contiguous int values. (This is similar to your homework, where you had a region of memory storing unsigned char values) The four lines shown here are legal. Which operation is illegal?

```
int *p1 = buf;
const int *p2 = buf;
int * const p3 = buf;
const int * p4 const = buf;


p2++;
*p1 = 3;
*p3 = 5;
p1++;
*p2 = 7
```

*p2 = 7;

[1311] The variable buf is a pointer to a region of memory storing contiguous int values. (This is similar to your homework, where you had a region of memory storing unsigned char values.) The four lines shown here are legal. Which operation is illegal?

```
int *p1 = buf;
const int *p2 = buf;
int * const p3 = buf;
const int * p4 const = buf;
```

p3++;

[1312] The variable buf is a pointer to a region of memory storing contiguous int values. (This is similar to your homework, where you had a region of memory storing unsigned char values.) The four lines shown here are legal. Which operation is legal?

```
int *p1 = buf;
const int *p2 = buf;
int * const p3 = buf;
const int * p4 const = buf;
```

*p3 = 5;

[1313] These pointer should point to "nothing". Which is not correctly initialized?

vector<int> *vp;

Star *ps = NULL;
vector<int> *vp(0);
int *pi = nullptr;
double *pd{};
All are correctly initialized to point to nothing	

[1315] Which of these is the preferred way to initialize a pointer so that it points to "nothing"?	int *pi = nullptr;
[1317] All of these are legal C++ statements; which of them uses the C++ address operator? int a = 3, b = 4;	int *p = &a;
[1318] All of these are legal C++ statements; which of them uses the C++ reference declarator? int a = 3, b = 4;	int &x = a;
[1319] All of these are legal C++ statements; which of them uses the C++ pointer declarator? int a = 3, b = 4;	int *p = &b;
[1320] All of these are legal C++ statements; which of them uses the C++ dereferencing operator? int a = 3, b = 4;	int x = *p;
[1321] All of these are legal C++ statements; which of them uses indirection? int a = 3, b = 4;	int x = *p;
[1322] In C++, global variables are stored:	in the static storage area
[1323] What is true about an uninitialized pointer?	Dereferencing it is undefined behavior
[1324] What is true about this code? int n{500}; int *p = &n;	*p is the value of n
[1325] What is true about this code? int * choice;	choice contains an undefined address
[1326] How can we print the address where n is located in memory? int n{500};	cout << &n << endl;

[1327] Which expression obtains the value that p points to?
int x(100);
int *p = &x;
	*p

[1328] What is a common pointer error?	Using a pointer without first initializing it
[1329] What is printed when you run this code? int x(100); cout << &x << endl;	The memory location where x is stored

[1330] What is printed when you run this code?
int n{};
int *p = &n;
*p = 10;
n = 20;
cout << *p << endl;
	20

[1331] What is printed when you run this code?
int num = 0;
int *ptr = #
num = 5;
*ptr += 5;
cout << num << " " << *ptr << endl;
	10 10

int *n{nullptr}; cout << n << endl;	
[1333] What is printed when you run this code? int *n{nullptr}; cout << *n << endl;	No compilation errors, but undefined behavior
[1334] What is printed when you run this code? int *n{nullptr}; cout << &n << endl;	The address value where n is stored
[1335] What is printed when you run this code? int *p = &0; cout << *p << endl;	No output; compiler error.
[1336] What is printed when you run this code? int n{}; int *p; *p = &n; cout << *p << endl;	Will not compile
[1337] What is printed when you run this code? int n{}; int *p; *p = n; cout << *p << endl;	No compilation errors, but undefined behavior when run
[1338] What is the term used to describe a variable with stores a memory address?	pointer
[1339] Which of these is not one of the three characteristics of every variable?	alias
[1340] Which area of memory is your program code stored in?	Text
[1341] Which area of memory are local variables stored in?	Stack
[1342] Which area of memory are global variables stored in?	Static storage area

[1343] Examine the following code. What is stored in c after it runs.
 	
int f(int * p, int x)
{
***p = x** * 2;
return x / 2;
}
. . .
int a = 3, b, c;
c = f(&b, a);
	1

[1344] Examine the following code. What is stored in b after it runs.
 	
int f(int * p, int x)
{
***p = x** * 2;
return x / 2;
}
. . .
int a = 3, b, c;
c = f(&b, a);
	6

[1345] Examine the following code. What is stored in a after it runs.
 	
int f(int * p, int x)
{
***p = x** * 2;
return x / 2;
}
. . .
int a = 3, b, c;
c = f(&b, a);
	3

[1346] Examine this version of the swap() function, which is different than the two versions appearing in your text. How do you call it?
 	
void swap(int& x, int * y)
{
. . .
}
. . .
int a = 3, b = 7;
// What goes here ?
	swap(a, &b);

void swap(int * x, int & y)
{
. . .
}
. . .
int a = 3, b = 7;
// What goes here ?	

[1348] Assume that p is a pointer to the first of 50 contiguous integers stored in memory. What is the address of the first integer appearing after this sequence of integers?	p + 50;
[1349] Assume that p1 is a pointer to an integer and p2 is a pointer to a second integer. Both integers appear inside a large contiguous sequence in memory, with p2 storing a larger address. How many total integers are there in the slice between p1 and p2?	p2 - p1;
[1350] Here is the pseudocode for the greenScreen() function in H12. What single statement sets the red, green and blue components to 0? Let p point the beginning of the image Set end to point just past the end While p != end If *(p + 3) is 0 (transparent) Clear all of the fields Increment p by 4	***(p) =** ***(p + 1) =** *(p + 2) = 0;
[1351] Here is a fragment of pseudocode for the negative() function in H12. What statement represents the underlined portion of code? Let p point to beginning of the image Let end be pixel one past the end of the image While p != end Invert the red component Move p to next component	p++;

Used to access the data inside a variable
 	
Determines the amount of memory required and the operations permitted on a variable
 	
The meaning assigned to a set of bits stored at a memory location
 	
An object whose value is an address in memory
 	
Expression using the address operator
 	
Expression using the reference declarator
 	
Expression using the dereferencing operator
 	
Expression using the pointer declarator
 	
Expression returning the number of allocated bytes used by an object
 	
Address value 0
	variable name
	
	variable type
	
	variable value
	
	pointer
	
	p = &a;
	
	int x = 3;
	
	y = *a;
	
	double * v;
	
	sizeof(Star)
	
	nullptr

[1401] Which of these lines correctly prints 3?
 	
struct S {
int a = 3;
double b = 2.5;
};
 	
S obj, *p = &obj;
 	
cout << p.a << endl;
cout << *p.a << endl;
cout << *(p).a << endl;
cout << *(p.a) << endl;
cout << (*p).a << endl;
	cout << (*p).a << endl;

[1402] Which of these lines correctly prints 2.5?
 	
struct S {
int a = 3;
double b = 2.5;
};
 	
S obj, *p = &obj;
 	
cout << *(p).b << endl;
cout << *p.b << endl;
cout << p->b << endl;
cout << *(p.b) << endl;
cout << *p->b << endl;
	cout << p->b << endl;

int a[15];
 	
cout << a[8] << endl;
cout << a(7) << endl;
cout << a.at(7) << endl;
cout << a[7] << endl;	

[1404] Which prints the number of elements in a?
 	
int a[] = {1, 2, 3};
 	
cout << a.length << endl;
cout << sizeof(a[0]) << endl;
cout << a.size() << endl;
cout << sizeof(a) << endl;
None of these
	None of these

[1405] What is stored in the last element of nums? int nums[3] = {1, 2}; Undefined value 2 Syntax error in array declaration 0 1	0
[1406] Which line throws and out_of_range exception? double speed[5] = {. . .}; None of these cout << speed[4] << endl; cout << speed[5] << endl; cout << speed[0] << endl; cout << speed[1] << endl;	None of these
[1407] Which line has undefined output? double speed[5] = {. . .}; cout << speed[5] << endl; cout << speed[0] << endl; None of these cout << speed[1] << endl; cout << speed[4] << endl;	cout << speed[5] << endl;
[1408] Which line creates an array with 5 elements? int[5] d; int b[5]; int a[4]; None of these int[] c[5];	int b[5];

[1409] What is printed?
 	
int a[] = {1, 2, 3};
int b[] = {1, 2, 3};
 	
if (a == b) cout << "a == b" << endl;
else cout << "a != b" << endl;
 	
a != b
Undefined behavior
a == b
Syntax error; does not compile.
	a != b

[1410] What does the array a contain after this runs? int a[] = {1, 2, 3}; int b[] = {4, 5, 6}; a = b; Syntax error; does not compile. {4, 5, 6} {1, 2, 3} Undefined behavior	Syntax error; does not compile.
[1411] Which assigns a value to the first position in letters? char letters[26]; letters[0] = 'a'; letters[0] = "a"; letters[1] = 'b'; letters.front() = 'a'; letters = 'a';	letters[0] = 'a';

char letters[26]; *letters = 'a'; *letters = "a"; *letters[0] = 'a'; *(letters + 1) = 'a'; *letters + 1 = 'b';	

[1413] What does this loop do?
 	
int a[] = {6, 1, 9, 5, 1, 2, 3};
int x(0);
for (auto e : a) x += e;
cout << x << endl;
 	
Counts the elements in a
Selects the largest value in a
Has no effect
Selects the smallest value in a
Sums the elements in a
	Sums the elements in a

[1414] What is the address of the first pixel in the last row of this image?
 	
Pixel *p; // address of pixel data
int w, h; // width and height of image
 	
p + w + h
p + w + (h - 1)
p + w * h
p + w * (h - 1)
None of these are correct
	p + w * (h - 1)

[1415] Which returns the last pixel on the first row of this image?
 	
Pixel *p; // address of pixel data
int w, h; // width and height of image
 	
*p + w - 1
None of these are correct
*(p + w) - 1
p + w - 1
*(p + w - 1)
	*(p + w - 1)

[1416] Which returns the last pixel on the first row of this image?
 	
Pixel *p; // address of pixel data
int w, h; // width and height of image
 	
p[w - 1]
*p[w - 1]
None of these are correct
p[w] - 1
p + w - 1
	p[w - 1]

[1417] What is the equivalent array notation? int dates[10]; cout << (*dates + 2) + 2 << endl; dates[0] + 4 dates[2] + 2 dates[2] dates[0] + 2 &dates[2]	dates[0] + 4
[1418] What is the equivalent array notation? int dates[10]; cout << (dates + 2) << endl; dates[2] + 2 &dates[2] dates[0] + 2 dates[2] dates[0] + 4	&dates[2]
[1419] What is the equivalent array notation? int dates[10]; cout << *(dates + 2) << endl; dates[2] + 2 dates[0] + 4 dates[2] &dates[2] dates[0] + 2	dates[2]

int dates[10];
cout << (*dates) + 2 << endl;

&dates[2]
dates[0] + 2
dates[0] + 4
dates[2]
dates[2] + 2

[1421] What is the equivalent array notation?

int dates[10];
cout << *dates + 2 << endl;

&dates[2]
dates[2] + 2
dates[0] + 4
dates[2]
dates[0] + 2

dates[0] + 2

[1422] What is the equivalent array notation?

int dates[10];
cout << *(dates + 2) + 2 << endl;

&dates[2]
dates[0] + 4
dates[0] + 2
dates[2]
dates[2] + 2

dates[2] + 2

[1423] What is the equivalent address-offset notation?

int a[] = {1, 2, 3, 4, 5, 6, 7};
int *p = a;

cout << a[1] * 2 << endl;

None of these
***p + 1** * 2
p + 1 * 2
(***p + 1)** * 2
***(p + 1)** * 2

***(p + 1)** * 2

[1424] What prints?

int a[] = {1, 3, 5, 7, 9};
int *p = a;
cout << *p++;
cout << *p << endl;

13
None of these
33
22
12

13

[1425] What prints?

int a[] = {1, 3, 5, 7, 9};
int *p = a;
cout << *++p;
cout << *p << endl;

33
13
None of these
22
12

33

[1426] What prints?

int a[] = {1, 3, 5, 7, 9};
int *p = a;
cout << ++*p;
cout << *p << endl;

13
12
None of these
22
33

22

int a[] = {1, 3, 5, 7, 9};
int *p3 = &a[1];
None of these
int *p1 = a;
int *p4 = &a;
int *p2 = a + 3;	

[1428] Which expression returns the number of countries? string countries[] = {"Andorra", "Albania", . . . }; len(countries) countries.length sizeof(countries) * sizeof(countries[0]) sizeof(countries) None of these	None of these
[1429] Which expression returns the number of countries? string countries[] = {"Andorra", "Albania", . . . }; sizeof(countries) len(countries) sizeof(countries) / sizeof(string) None of these sizeof(countries) * sizeof(countries[0])	sizeof(countries) / sizeof(string)
[1430] Which expression returns the number of countries? string countries[] = {"Andorra", "Albania", . . . }; len(countries) sizeof(countries) * sizeof(countries[0]) sizeof(countries) None of these sizeof(countries) / sizeof(countries[0])	sizeof(countries) / sizeof(countries[0])
[1431] Which array definition is illegal? int SIZE = 3; int a1[SIZE]; int a2[3]; int a3[3]{}; int a4[] = {1, 2, 3}; int a5[3] = {1, 2}; a2 a3 None of these a1 a5	a1
[1432] Which array definition contains undefined values? int SIZE = 3; int a1[SIZE]; int a2[3]; int a3[3]{}; int a4[] = {1, 2, 3}; int a5[3] = {1, 2}; a3 a1 None of these a5 a2	a2
[1433] Which array definition is initialized to all zeros? int SIZE = 3; int a1[SIZE]; int a2[3]; int a3[3]{}; int a4[] = {1, 2, 3}; int a5[3] = {1, 2}; a5 a2 None of these a3 a1	a3

int SIZE = 3;
int a1[SIZE];
int a2[3];
int a3[3]{};
int a4[] = {1, 2, 3};
int a5[3] = {1, 2};

a5
a3
None of these
a2
a1

[1435] Which array definition is illegal?

const int SIZE = 3;
int a1[SIZE];
int a2[3];
int a3[3]{};
int a4[] = {1, 2, 3};
int a5[2] = {1, 2, 3};
a2
a5
a3
None of these
a1

a5

[1436] Which array definition produces {1, 2, 0}?

int SIZE = 3;
int a1[SIZE];
int a2[3];
int a3[3]{};
int a4[] = {1, 2, 3};
int a5[3] = {1, 2};

a3
a5
a2
a1
None of these

a5

In C++ using == to compare one array to another is permitted (if meaningless).

You must use an integral constant or literal to specify the size of a built-in C++ array.

The reinterpret_cast instruction changes way that a pointer's indirect value is interpreted.

If p is a pointer to a structure, and the structure contains a data member x, you can access the data member by using the notation: (*p).x

C++ arrays have no support for bound-checking.

In C++ assigning one array to another is illegal

The allocated size of a built-in C++ array cannot be changed during runtime.

The size of the array is not stored along with its elements.

If img is a pointer to the first byte in an image loaded into memory, Pixel is a structure as defined in your textbook, you can create a Pixel pointer pointing to the image by writing:
Pixel **p = reinterpret_cast<Pixel** >(img);

The subscripts of a C++ array range from 0 to the array size - 1.

C++ arrays have no built-in functions for inserting and deleting.

A forward reference can be used when you want to use a pointer to a structure as a data member without first defining the entire structure.

The elements of a C++ array created in a function are allocated on the stack.

The elements of a C++ array created outside of a function are allocated in the static-storage area.

The elements of a C++ string array with no explicit initialization, created in a function will be set to the empty string.

Explicitly initializing an array like this: int a[3] = {1, 2, 3}; requires the size to be the same or larger than the number of elements supplied.

In C++ printing an array name prints the address of the first element in the array.

In C++ there is no separate array variable. The array name is a symbolic representation of the address of the first element in the array.

In C++ initializing an array with the contents of another is illegal.

C++ arrays produce undefined results if you access an element outside the array.

Explicitly initializing an array like this: int a[] = {1, 2, 3}; works in all versions of C++.

You may use any kind of integral variable to specify the size of a built-in C++ array.

The elements of a C++ string array with no explicit initialization, created in a function will be set to null.

Explicitly initializing an array like this: int a[3] = {1, 2, 3}; requires the size to be the same or smaller than the number of elements supplied.

In C++ using == to compare one array to another is illegal.

The allocated size of a built-in C++ array may be changed during runtime

If img is a pointer to the first byte in an image loaded into memory, Pixel is a structure as defined in your textbook, you can create a Pixel pointer pointing to the image by writing:
Pixel **p = static_cast<Pixel** >(img);

The reinterpret_cast instruction produces a temporary value by converting its argument.

In C++ initializing an array with the contents of another is permitted.

C++ arrays use bound-checking when you access their elements with the at() member function.

The elements of a C++ array created in a function are allocated on the heap.

In C++ assigning one array to another is permitted.

C++ arrays throw an out_of_bounds exception if you access an element outside the array.

In C++ an array variable and the array elements are separate. The array variable contains the address of the first element in the array.

In C++ printing an array name prints the value of the first element in the array.

The elements of a C++ int array with no explicit initialization, created in a function will be set to zero.

C++ arrays can be allocated with a size of 0.

The static_cast instruction changes way that a pointer's indirect value is interpreted.

The size of the array is stored along with its elements.

The allocated size of a built-in C++ array may be changed during runtime

A forward reference can be used when you want to use a structure as a data member without first defining the entire structure.

The elements of a C++ array created outside of a function are allocated on the stack.

If p is a pointer to a structure, and the structure contains a data member x, you can access the data member by using the notation: *p->x

C++ arrays offer built-in member functions for inserting and deleting.

Explicitly initializing an array like this: int a[] = {1, 2, 3}; only works in C++ 11.

[1501] Below is a cumulative algorithm using an array and a range-based loop. What is printed? (Assume this is inside main() with all includes, etc.)

```
int a[] = {2, 4, 6, 8};
int sum = 0;
for (auto e : a) sum += e;
cout << "sum->" << sum << endl;
```

Compiles but crashes with an endless loop.
Does not compile. Cannot use range-loop on arrays.
sum->20
sum->0
Compiles and runs, but results are undefined.

sum->20

[1502] Below is a cumulative algorithm using an array and a range-based loop. What is printed? (Assume this is inside main() with all includes, etc.)

```
int a[] = {2, 4, 6, 8};
int sum;
for (auto e : a) sum += e;
cout << "sum->" << sum << endl;
```

Compiles and runs, but results are undefined.
sum->20
sum->8
Does not compile. Cannot use range-loop on arrays.
Compiles but crashes with an endless loop.

Compiles and runs, but results are undefined.

int a[] = {2, 4, 6, 8};
int sum = 0;
for (auto e : a) sum += e;
cout << "sum->" << e << endl;

Does not compile; e is undefined.
Does not compile. Cannot use range-loop on arrays.
Compiles and runs, but results are undefined.
sum->20
sum->8

[1504] Below is a cumulative algorithm using an array and a range-based loop. What is printed? (Assume this is inside main() with all includes, etc.)

int a[] = {2, 4, 6, 8};
int sum = 0;
for (auto e : a) sum =+ e;
cout << "sum->" << sum << endl;
Does not compile. Cannot use range-loop on arrays.
sum->8
Compiles and runs, but results are undefined.
sum->20
Does not compile; e is undefined.

sum->8

[1505] Below is a cumulative algorithm using an array and an iterator-based loop. What is printed? (Assume all includes have been added, etc.)

double average(const int ***beg, const int** *end)
{
double sum = 0;
size_t count = end - beg;
while (beg != end) sum += *beg++;
return sum / count;
}

int main()
{
int a[] = {2, 4, 6, 8};
cout << average(begin(a), end(a)) << endl;
}

4
5
Does not compile
6
Endless loop when run; likely crashes.

5

[1506] Below is a cumulative algorithm using an array and an iterator-based loop. What is printed? (Assume all includes have been added, etc.)

double average(const int **beg, const int** end)
{
double sum = 0;
size_t count = end - beg;
while (beg != end) sum += *beg++;
return sum / count;
}

int main()
{
int a[] = {2, 4, 6, 8};
cout << average(begin(a), end(a) - 1) << endl;
}
Endless loop when run; likely crashes.
Does not compile
4
5
6

4

double average(const int **beg, const int** end)
{
double sum = 0;
size_t count = end - beg;
while (beg != end) sum += *beg++;
return sum / count;
}

int main()
{
int a[] = {2, 4, 6, 8};
cout << average(begin(a) + 1, end(a)) << endl;
}

6
4
5
Does not compile
Endless loop when run; likely crashes.

[1508] Below is a cumulative algorithm using an array and an iterator-based loop. What is printed? (Assume all includes have been added, etc.)

double average(const int **beg, const int** end)
{
double sum = 0;
size_t count = end - beg;
while (beg != end) sum += *beg++;
return sum / count;
}

int main()
{
int a[] = {2, 4, 6, 8};
cout << average(end(a), begin(a)) << endl;
}

Does not compile
Endless loop when run; likely crashes.
5
6
4

Endless loop when run; likely crashes.

[1509] Below is a cumulative algorithm using an array and an iterator-based loop. What is printed? (Assume all includes have been added, etc.)

double average(const int **beg, const int** end)
{
if (end <= beg) return 0.0 / 0.0; // nan
double sum = 0;
size_t count = end - beg;
while (beg != end) sum += *beg++;
return sum / count;
}

int main()
{
int a[] = {2, 4, 6, 8};
cout << average(end(a), begin(a)) << endl;
}

4
Does not compile
5
Not a number (NaN)
Endless loop when run; likely crashes.

Not a number (NaN)

double average(const int **beg, const int** end)
{
double sum = 0;
size_t count = end - beg;
while (beg != end) sum += *beg++;
return sum / count;
}

int main()
{
int a[] = {2, 4, 6, 8};
cout << average(a, a + 1) << endl;
}

Does not compile
3
2
5
4

[1511] Below is a cumulative algorithm using an array and an iterator-based loop. What is printed? (Assume all includes have been added, etc.)

double average(const int **beg, const int** end)
{
double sum = 0;
size_t count = end - beg;
while (beg != end) sum += *beg++;
return sum / count;
}

int main()
{
int a[] = {2, 4, 6, 8};
cout << average(a, a + 2) << endl;
}

Does not compile
5
3
4
2

3

[1512] Below is a cumulative algorithm using an array and an iterator-based loop. What is printed? (Assume all includes have been added, etc.)

double average(const int **beg, const int** end)
{
double sum = 0;
size_t count = end - beg;
while (beg != end) sum += *beg++;
return sum / count;
}

int main()
{
int a[] = {2, 4, 6, 8};
cout << average(a + 1, a + 3) << endl;
}
5
2
Does not compile
4
3

5

[1513] Below is a cumulative algorithm using an array and an iterator-based loop. What is printed? (Assume all includes have been added, etc.)

double average(const int **beg, const int** end)
{
double sum = 0;
size_t count = end - beg;
while (beg != end) sum += *beg++;
return sum / count;
}

int main()
{
int a[] = {2, 4, 6, 8};
cout << average(a, a + 3) << endl;
}
5
Does not compile
4
2
3

4

const int a[] = {2, 4, 6, 8};
cout << mystery(a, 4) << endl;

void mystery(const int a[], size_t n);
int mystery(int a[], size_t n);
int mystery(const int a*, size_t n);
int mystery(const int *a, size_t n);
int mystery(const int[] a, size_t n);

[1515] What is the correct prototype for mystery? (It may modify the array.)

const int a[] = {2, 4, 6, 8};
cout << mystery(a, 4) << endl;

int mystery(int[] a, size_t n);
int mystery(int a, size_t n);
int mystery(int *a, size_t n);
int mystery(int a*, size_t n);
void mystery(const int a[], size_t n);

int mystery(int *a, size_t n);

[1516] What is printed here? (Assume all includes have been added. Assume 4-bytes per int, 8 bytes per pointer.)

size_t len(const int a[])
{
return sizeof(a) / sizeof(a[0]);
}

int main()
{
int a[] = {2, 4, 6, 8};
cout << len(a) << endl;
}
2
Does not compile
1
4

2

[1517] What is printed here? (Assume all includes have been added. Assume 4-bytes per int, 8 bytes per pointer.)

int main()
{
int a[] = {2, 4, 6, 8};
cout << sizeof(a) / sizeof(a[0]) << endl;
}

Does not compile
4
1
2

4

[1518] What is printed here? (Assume all includes have been added. Assume 4-bytes per int, 8 bytes per pointer.)

size_t len(const int **a, const int** b)
{
return b - a;
}

int main()
{
int a[] = {2, 4, 6, 8};
cout << len(begin(a), end(a)) << endl;
}
Does not compile
4
2
1

4

[1519] What is printed here? (Assume all includes have been added. Assume 4-bytes per int, 8 bytes per pointer.)

size_t len(const int **a, const int** b)
{
return b - a;
}

int main()
{
int a[] = {2, 4, 6, 8};
cout << len(a, a + 3) << endl;
}
2
3
4
Does not compile

3

```
int odds(int a[], size_t len)
{
int sum = 0;
for (size_t i = 0; i < len; i++)
if (a[i] % 2 == 1) sum += a[i]++;
return sum;
}

int main()
{
int a[] = {1, 3, 5};
cout << odds(a, 3) << odds(a, 2)
<< odds(a, 1) << endl;

}
```

999
900
300
941
Does not compile

[1521] What does this function do?

```
int mystery(const int a[], size_t n)
{
int x = n - 1;
while (n > 0)
{
n--;
if (a[n] > a[x]) x = n;
}
return x;
}
```

Returns the largest number in the array
Returns the index of the last occurrence of the largest number in the array
Returns the smallest number in the array
Returns the index of the first occurrence of the largest number in the array
Does not compile

Returns the index of the last occurrence of the largest number in the array

[1522] What does this function do?

```
int mystery(const int a[], size_t n)
{
int x = n - 1;
while (n > 0)
{
n--;
if (a[n] < a[x]) x = n;
}
return x;
}
```

Returns the smallest number in the array
Returns the index of the last occurrence of the smallest number in the array
Does not compile
Returns the index of the first occurrence of the smallest number in the array
Returns the largest number in the array

Returns the index of the last occurrence of the smallest number in the array

[1523] What does this function do?

```
int mystery(const int a[], size_t n)
{
int x = a[n - 1];
while (n > 0)
{
n--;
if (a[n] < a[x]) x = a[n];
}
return x;
}
```

Returns the index of the first occurrence of the smallest number in the array
Returns the largest number in the array
Returns the index of the last occurrence of the smallest number in the array
Returns the smallest number in the array
Does not compile

Returns the smallest number in the array

int mystery(const int a[], size_t n)
{
int x = a[n - 1];
while (n > 0)
{
n--;
if (a[n] > a[x]) x = a[n];
}
return x;
}
Returns the index of the last occurrence of the smallest number in the array
Does not compile
Returns the largest number in the array
Returns the smallest number in the array
Returns the index of the first occurrence of the smallest number in the array

[1525] What is printed?

int mystery(const int a[], size_t n)
{
int x = a[n - 1];
while (n > 0)
{
n--;
if (a[n] > a[x]) x = a[n];
}
return x;
}

int main()
{
int a[] = {1, 3, 5, 3, 5, 4};
cout << mystery(a, 6) << endl;
}

5

[1526] What is printed?

int mystery(const int a[], size_t n)
{
int x = n - 1;
while (n > 0)
{
n--;
if (a[n] < a[x]) x = n;
}
return x;
}

int main()
{
int a[] = {1, 2, 5, 2, 5, 4};
cout << mystery(a, 6) << endl;
}
1
2
3
None of these
4

None of these

[1527] What is printed?

int mystery(const int a[], size_t n)
{
int x = n - 1;
while (n > 0)
{
n--;
if (a[n] < a[x]) x = n;
}
return x;
}

int main()
{
int a[] = {4, 2, 5, 2, 5, 4};
cout << mystery(a, 6) << endl;
}

1
4
None of these
3
2

3

```
int mystery(const int a[], size_t n)
{
int x = n - 1;
while (n > 0)
{
n--;
if (a[n] > a[x]) x = n;
}
return x;
}

int main()
{
int a[] = {4, 2, 5, 2, 5, 4};
cout << mystery(a, 6) << endl;
}
```

None of these
4
3
1
2

[1529] What is printed?

```
int mystery(const int a[], size_t n)
{
int x = 0;
for (size_t i = 0; i < n; i++)
if (a[i] > a[x]) x = i;

return x;
}

int main()
{
int a[] = {4, 2, 5, 2, 5, 4};
cout << mystery(a, 6) << endl;
}
```

5
None of these
0
2
4

2

[1530] What is printed?

```
int mystery(const int a[], size_t n)
{
int x = 0;
for (size_t i = 0; i < n; i++)
if (a[i] < a[x]) x = i;

return x;
}

int main()
{
int a[] = {4, 2, 5, 2, 5, 4};
cout << mystery(a, 6) << endl;
}
```

None of these
2
0
1
3

1

[1531] What is printed?

```
const int mystery(const int p, size_t n)
{
const int x = p, y = p + n;
while (++p != y) {
if (p > x) x = p;
}
return x;
}

int main()
{
int a[] = {1, 2, 3, 4, 5, 1};
cout << *(mystery(a, 6)) << endl;
}
```

0
5
2
None of these
4

5

```
const int mystery(const int p, size_t n)
{
const int x = p, y = p + n;
while (++p != y) {
if (p > x) x = p;
}
return x;
}

int main()
{
int a[] = {1, 2, 3, 4, 5, 1};
cout << *(mystery(a, 6)) << endl;
}
```

4
5
2
None of these
0

[1533] What does this function do?

```
double mystery(const double a[], size_t len)
{
double x = a[0];
for (size_t i = 1; i < len; i++)
if (a[i] > x) x = a[i];
return x;
}
```

Does not compile
Returns the largest number in the array
Returns the smallest number in the array
Undefined. Depends on the input.

Returns the largest number in the array

[1534] What does this function do?

```
double mystery(const double a[], size_t len)
{
double x = a[0];
for (size_t i = 1; i < len; i++)
if (a[i] < x) x = a[i];
return x;
}
```

Returns the largest number in the array
Does not compile
Returns the smallest number in the array
Undefined. Depends on the input.

Undefined. Depends on the input.

[1535] What does this function do?

```
double mystery(const double a[], size_t len)
{
double x = 0;
for (size_t i = 0; i < len; i++)
if (a[i] > x) x = a[i];
return x;
}
```

Undefined. Depends on the input.
Does not compile
Returns the largest number in the array
Returns the smallest number in the array

Undefined. Depends on the input.

[1536] What does this function do?

```
double mystery(const double a[], size_t len)
{
double x = 0;
for (size_t i = 0; i < len; i++)
if (a[i] < x) x = a[i];
return x;
}
```

Returns the largest number in the array
Returns the smallest number in the array
Undefined. Depends on the input.
Does not compile

template <typename T>
ostream& mystery(ostream& out, const T* p, size_t n)
{
out << '[';
if (n) {
out << p[0];
for (size_t i = 1; i < n; i++)
out << ", " << p[i];
}
out << "]";
return out;
}

A cumulative algorithm
An extreme values algorithm
An iterator algorithm
None of these
A fencepost algorithm

[1538] What is printed?

template <typename T>
ostream& mystery(ostream& out, const T* p, size_t n)
{
out << '[';
if (n) {
out << p[0];
for (size_t i = 1; i < n; i++)
out << ", " << p[i];
}
out << "]";
return out;
}

int a[] = {1,2,3,4,5,1};
mystery(cout, a, 4) << endl;
[1, 2, 3]
[1, 2, 3, 4, 5, 1]
None of these or undefined output.
[1, 2, 3, 4, 5]
[1, 2, 3, 4]

[1, 2, 3, 4]

[1539] What is printed?

template <typename T>
ostream& mystery(ostream& out, const T* p, size_t n)
{
out << '[';
if (n) {
out << p[0];
for (size_t i = 1; i < n; i++)
out << ", " << p[i];
}
out << "]";
return out;
}

int a[] = {1,2,3,4,5,1};
mystery(cout, a, sizeof(a)) << endl;
[1, 2, 3, 4, 5, 1]
[1, 2, 3, 4]
[1, 2, 3, 4, 5]
None of these or undefined output.
[1, 2, 3]

None of these or undefined output.

[1540] What is printed?

template <typename T>
ostream& mystery(ostream& out, const T* p, size_t n)
{
out << '[';
if (n) {
out << p[0];
for (size_t i = 1; i < n; i++)
out << ", " << p[i];
}
out << "]";
return out;
}
. . .
int a[] = {1,2,3,4,5,1};

mystery(cout, a, sizeof(a) / sizeof(a[0])) << endl;
None of these or undefined output.
[1, 2, 3, 4]
[1, 2, 3]
[1, 2, 3, 4, 5]
[1, 2, 3, 4, 5, 1]

[1, 2, 3, 4, 5, 1]

```
template <typename T>
ostream& mystery(ostream& out, const T* p, size_t n)
{
out << '[';
if (n) {
out << p[0];
for (size_t i = 1; i < n; i++)
out << ", " << p[i];
}
out << "]";
return out;
}
. . .
int a[] = {1,2,3,4,5,1};

mystery(cout, a, 0)) << endl;
```

[0]
Does not compile. Arrays cannot be 0 length.
[]
[1]
No output

Elements always allocated on the heap	vector
How arrays are passed to functions	by address
What happens to an array when passed to a function	decays
const int *array	Elements may not be modified; pointer may be
int * const array	Elements in may be modified; pointer may not
const int * const array	Neither pointer nor elements in may be modified
sizeof(a) / sizeof(a[0])	Elements in array using arithmetic
end(a) - begin(a)	Elements in array using pointer difference
for (auto e : a) . .	A range-based loop
x = 0; for (auto e : a) x += e;	Cumulative algorithm
x = a[0]; for (auto e: a) if (e > x) x = e;	Extreme values algorithm
auto p = a; while (p != end(a)) p++;	Iterator-based loop
cout << a[0]; while (i < len) cout << ", " << a[i++];	Fence-post algorithm

An array passed to a function decays to a pointer.

An array passed to a function f(int * const a, ...) may have its elements changed.

The elements of an array may be allocated on the stack.

If p points to the first element in [1, 3, 5] then cout << ++*p prints 2.

The library function begin(a) returns a pointer to the first element in the array a.

The elements of an array may be allocated in the static storage area.

Arrays generally have higher performance than a vector.

The function mystery(const int**, const int**) likely employs an iterator loop.

The expression begin(a) + 1 returns a pointer to the second element in the array a.

Array subscripts are not range checked

An array passed to a function is passed by address.

If size_t len = 0; then len - 1 is the largest possible unsigned number.

If p points to the first element in [1, 3, 5] then cout << *++p prints 3.

The algorithm that finds the address of the smallest element in an array is called an extreme values algorithm.

The expression **p++ means the same as** (p++).

Before passing an array to a function, sizeof(a)/sizeof(a[0]) will tell the number of elements in the array.

For systems programming (such as operating systems), arrays are used more often than vectors.

The library function end(a) returns a pointer to position right past the last element in the array a.

For embedded systems, arrays are preferred over vector.

The parameter declarations int *p and int p[] mean the same thing.

The algorithm that prints elements separated by commas is called the fencepost algorithm.

The elements of a vector are allocated on the heap.

A vector variable may be allocated on the stack.

Before passing an array to a function, sizeof(a) will tell you the array's allocated size, but not the number of elements.

After passing an array to a function, sizeof(a)/sizeof(a[0]) will tell the number of elements in the array.

If p points to the first element in [1, 3, 5] then cout << *++p prints 1.

If p points to the first element in [1, 3, 5] then cout << ++*p prints 1.

The library function begin(a) returns a pointer to the element right before the first in the array a.

For embedded systems, vector is preferred over arrays.

For systems programming (such as operating systems), vectors are used more often than arrays.

For an equivalent number of elements, a vector will use less memory than an array.

The expression **p++ means the same as (**p)++.

An array passed to a function f(const int *a, ...) may have its elements changed.

The elements of a vector may be allocated on the stack.

For an equivalent number of elements, a vector will use more memory than an array.

The algorithm that prints elements separated by commas is called a cumulative algorithm.

The algorithm that finds the position of the largest element in an array is called a cumulative algorithm.

The algorithm that finds the position of the largest element in an array is called a cumulative algorithm.

After passing an array to a function, sizeof(a) will tell you the array's allocated size, but not the number of elements.

If p points to the first element in [1, 3, 5] then cout << *p++ prints 3.

The library function end(a) returns a pointer to the last element in the array a.

A vector generally has higher performance than an array.

If size_t len = 0; then len - 1 is the smallest possible unsigned number.

The expression begin(a) + 1 returns a pointer to the first element in the array a.

The function mystery(const int**, const int**) likely employs a counter-controlled loop.

An array passed to a function is passed by reference.

[1601] Below is a partially-filled array. If you are adding elements to this array in a loop, what is the correct loop bounds condition?

```
const size_t MAX = 100;
double nums[MAX];
size_t size = 0;
```

while (MAX < size) . . .
while (size < MAX) . . .
while (size <= MAX) . . .
for (size = 0; size < MAX; size++) . . .

while (size < MAX) . . .

[1602] Below is a partially-filled array. When adding elements to this array in a loop, what statement(s) correctly updates the array with value?

```
const size_t MAX = 100;
double nums[MAX];
size_t size = 0;
double value;
```

nums[size] = value;
nums[size++] = value;
nums[++size] = value;
size++; nums[size] = value;

nums[size++] = value;

[1603] Below is a partially-filled array. If you have a sentinel loop where the sentinel is a negative number, which of these conditions correctly reads the number named value?

```
const size_t MAX = 100;
double nums[MAX];
size_t size = 0;
double value;
```

cin >> value; if (value < 0) break;
if (! (cin >> value) || value < 0) break;
cin >> value; if (cin.fail() && value < 0) break;
if (value >= 0 && cin >> value) . . . // process value

if (! (cin >> value) || value < 0) break;

const size_t MAX = 100;
double nums[MAX];
size_t size = 0;
 	
double& back(double a[], size_t size);
 	
double& back(double a[], size_t& size);
 	
double& back(const double a[], size_t& size);
 	
double& back(double a[], size_t size, size_t MAX);	

[1605] Below is a declaration for a partially-filled array. What is the correct prototype for a function add() that appends a new element to the end of the array and returns true if successful?
 	
const size_t MAX = 100;
double nums[MAX];
size_t size = 0;
 	
bool add(double a[], size_t MAX, double e);
 	
bool add(double a[], size_t& size, double e);
 	
bool add(double a[], size_t size, size_t MAX, double e);
 	
bool add(double a[], size_t& size, size_t MAX, double e);
	bool add(double a[], size_t& size, size_t MAX, double e);

[1606] Below is a declaration for a partially-filled array. What is the correct prototype for a function insert() that inserts a new element at position pos in the array, shifts the remaining elements right, and returns true if successful?
 	
const size_t MAX = 100;
double nums[MAX];
size_t size = 0;
 	
bool insert(double a[], size_t& size, double e, size_t pos);
 	
bool insert(double a[], size_t MAX, double e, size_t pos);
 	
bool insert(double a[], size_t size, size_t MAX, double e, size_t pos);
 	
bool insert(double a[], size_t& size, size_t MAX, double e, size_t pos);
	bool insert(double a[], size_t& size, size_t MAX, double e, size_t pos);

[1607] Below is a declaration for a partially-filled array. What is the correct prototype for a function delete() that deletes the element at position pos in the array, shifts the remaining elements left, and returns true if successful?
 	
const size_t MAX = 100;
double nums[MAX];
size_t size = 0;
 	
bool delete(double a[], size_t size, size_t pos);
 	
bool delete(double a[], size_t& size, size_t pos);
 	
bool delete(double a[], size_t MAX, size_t& pos);
 	
bool delete(const double a[], size_t& size, size_t pos);
	bool delete(double a[], size_t& size, size_t pos);

[1608] Below is a mystery() function with no types for its parameter. What does the function do?
 	
void mystery(a, b&, c, d, e)
{
b = 0;
while (in >> n && b < c)
a[b++] = n;
}
 	
Inserts input into a partially-filled array
Deletes elements from a partially-filled array
Appends input to the end of a partially-filled array.
	Appends input to the end of a partially-filled array.

void mystery(a, b&, c, d, e)
{
for (i = d; i < b; i++)
a[i] = a[i + 1];
b--;
}
 	
Inserts input into a partially-filled array
Deletes elements from a partially-filled array
Appends input to the end of a partially-filled array.	

[1610] Below is a mystery() function with no types for its parameter. What does the function do?
 	
void mystery(a, b&, c, d, e)
{
for (i = b; i > d; i--)
a[i] = a[i - 1];
a[d] = e;
b++;
}
 	
Inserts input into a partially-filled array
Deletes elements from a partially-filled array
Appends input to the end of a partially-filled array.
	Inserts input into a partially-filled array

[1611] Below is a template function, push(), that adds elements to the end of a partially-filled array, returning true if successful. The function has an error; what is the error?
 	
template <typename T>
bool push(T* a, size_t& size, size_t MAX, T e)
{
if (size < MAX) {
a[size] = e;
return true;
}
return false;
}
 	
a should be a const T*
size should be incremented
size should be passed by value
Condition should be size <= MAX
	size should be incremented

[1612] Below is pop(), a template function that works with a partially-filled array. The function copies the last element in the array into the output parameter e and returns true if successful; it returns false otherwise. What is the error?
 	
template <typename T>
bool pop(T* a, size_t& size, T& e)
{
if (size) {
e = a[size];
size--;
return true;
}
return false;
}
 	
a should be a const T*
Condition should be !size
size should be incremented
The wrong value is assigned to e
	The wrong value is assigned to e

[1613] Below is index(), a template function that works with a partially-filled array. The function searches the array a for the value e and returns its position. It returns NOT_FOUND if the value does not it exist in the array. The function contains an error; what is the error?
 	
const size_t NOT_FOUND = static_cast<size_t>(-1);
template <typename T>
size_t index(const T* a, size_t& size, T e)
{
for (size_t i = 0; i < size; i++)
if (a[i] == e) return i;
return NOT_FOUND;
}
 	
a should not be a const T*
e should be passed by reference
The condition should go to i <= size
size should not be passed by reference
	size should not be passed by reference

removed. The function contains an error; what is the error?

```
template <typename T>
int remove(T* a, size_t& size, T e)
{
int removed = 0;
size_t i = 0;
while (i < size)
{
if (a[i] == e)
{
removed++;
size--;
for (size_t j = i; i < size; i++)
a[i] = a[i + 1];
}
i++;
}
return removed;
}
```

a should be a const T*
size should not be passed by reference
The condition should go to while (i <= size)
Not all copies of e are necessarily removed

[1615] Below is insert(), a template function that works with a partially-filled array. The function inserts the argument e into the array, in sorted order. The function returns true if it succeeds, false otherwise. The function contains an error; what is the error?

```
template <typename T>
bool insert(T* a, size_t& size, size_t MAX, T e)
{
if (size < MAX) return false;
size_t i = 0;
while (i < size)
{
if (a[i] > e) break;
i++;
}
for (j = size; j > i; j--)
a[j] = a[j - 1];
a[i] = e;
size++;

return true;
}
```

The value is inserted into the wrong position

The second loop should start at i and go up to size

When a value is inserted, it erases one of the existing values

If there is room to insert, the function returns false instead of true

---

[1616] Below is insert(), a template function that works with a partially-filled array. The function inserts the argument e into the array, in sorted order. The function returns true if it succeeds, false otherwise. The function contains an error; what is the error?

```
template <typename T>
bool insert(T* a, size_t& size, size_t MAX, T e)
{
if (size < MAX) return false;
size_t i = 0;
while (i < size)
{
if (a[i] > e) break;
i++;
}
for (j = size; j > i; j--)
a[j] = a[j - 1];
a[i] = e;
size++;


return true;
}
```

The value is inserted into the wrong position

The second loop should start at i and go up to size

When a value is inserted, it erases one of the existing values

If the array is full, the function overwrites memory outside the array

true if it succeeds, false otherwise. The function contains an error; what is the error?
 	
template <typename T>
bool insert(T* a, size_t& size, size_t MAX, T e)
{
if (size >= MAX) return false;
size_t i = 0;
while (i < size)
{
if (a[i] > e) break;
i++;
}
for (j = size; j > i; j--)
a[j] = a[j - 1];
a[i] = e;
 	
return true;
}
 	
The value is inserted into the wrong position
 	
The second loop should start at i and go up to size
 	
Every time the function is called, an array element is "lost"
 	
The function writes over memory outside the array when it should not	

[1618] Which loop is used when inserting an element into an array?
 	
for (j = pos; j < size; j++) a[j] = a[j + 1];
for (j = size; j > pos; j--) a[j] = a[j - 1];
for (j = MAX; j > size; j--) a[j - 1] = a[j];
for (j = size; j < MAX; j++) a[j - 1] = a[j];
	for (j = size; j > pos; j--) a[j] = a[j - 1];

[1619] Which loop is used when deleting an element from an array?
 	
for (j = MAX; j > size; j--) a[j - 1] = a[j];
for (j = pos; j < size; j++) a[j] = a[j + 1];
for (j = size; j > pos; j--) a[j] = a[j - 1];
for (j = size; j < MAX; j++) a[j - 1] = a[j];
	for (j = pos; j < size; j++) a[j] = a[j + 1];

[1620] Assume you have a partially filled array a, with variables size and MAX (capacity). To append value to the array, which of these assignments is correct?
 	
a[size] = value;
a[size + 1] = value;
a[size - 1] = value;
a[MAX - 1] = value;
	a[size] = value;

[1621] Below is startsWith(), a template function that works with two partially-filled arrays. The function returns true if the array a "starts with" the same elements as the array b, false otherwise. The function contains an error; what is the error?
 	
template <typename T>
bool startsWith(const T* **a, size_t sizeA, const T*** b, size_t sizeB)
{
if (sizeA > sizeB) return false;
for (size_t i = 0; i < sizeB; i++)
if (a[i] != b[i]) return false;
return true;
}
 	
The condition i < sizeB should be i <= sizeB
The condition a[i] != b[i] should be b[i] == a[i]
sizeA and sizeB should both be passed by reference
The condition (sizeA > sizeB) should be (sizeB > sizeA)
	The condition (sizeA > sizeB) should be (sizeB > sizeA)

[1622] Below is endsWith(), a template function that works with two partially-filled arrays. The function returns true if the array a "ends with" the same elements as the array b, false otherwise. The function contains an error; what is the error?
 	
template <typename T>
bool endsWith(T* **a, size_t sizeA, T*** b, size_t sizeB)
{
if (sizeA < sizeB) return false;
size_t diff = sizeA - sizeB;
for (size_t i = 0; i < sizeB; i++)
if (a[i + diff] != b[i]) return false;
return true;
}
 	
The arrays a and b should be const T*
sizeA and sizeB should both be passed by reference
The condition (sizeA < sizeB) should be (sizeA > sizeB)
The condition a[i + diff] != b[i] should be a[i - diff] == b[i]
	The arrays a and b should be const T*

function contains an error; what is the error?

```
template <typename T>
int removeDupes(T* a, size_t& size)
{
int count = 0;
for (size_t i = 0; i < size; i++) {
for (size_t j = i + 1; j < size; j++) {
if (a[i] == a[j]) { // duplicate
size--; count++;
for (size_t k = j; k < size; k++)
a[k] = a[k + 1];
}
}
}
return count;
}
```

The array parameter should be const T

It removes some duplicates, but not all of them

It returns a different number than the actual elements removed

It produces undefined behavior by exceeding the bounds of the array

In a partially-filled array, the capacity may be less than the array's size.

When inserting a value into a partially-filled array, in ascending order, the insertion position may be the same as capacity.

When inserting elements into a partially-filled array, the array should be declared const.

When comparing two partially-filled arrays for equality, both arrays should not be declared const.

When deleting an element from a partially-filled array, it is an error if the index of the element to be removed is < size.

When inserting a value into a partially-filled array, elements following the insertion position are shifted to the left.

In a partially-filled array, the size represents the allocated size of the array.

In a partially-filled array, the capacity represents the effective size of the array.

In a partially-filled array, all of the elements are not required to contain meaningful values

When inserting an element into a partially-filled array, it is an error if size < capacity.

In a partially-filled array, all of the elements contain meaningful values

When deleting elements from a partially-filled array, the array should be declared const.

In a partially-filled array capacity represents the number of elements that are in use.

When searching for the index of a particular value in a partially-filled array, the array should not be declared const.

When inserting a value into a partially-filled array, in ascending order, the insertion position is the index of the first value smaller than the value.

False

When inserting a value into a partially-filled array, in ascending order, the insertion position may be the same as size.

When inserting a value into a partially-filled array, in descending order, the insertion position is the index of the first value smaller than the value.

When removing an element from a partially-filled array, elements following the deletion position are shifted to the left.

When deleting elements from a partially-filled array, the array should not be declared const.

In a partially-filled array size represents the number of elements that are in use.

When inserting a value into a partially-filled array, elements following the insertion position are shifted to the right.

In a partially-filled array, the capacity represents the allocated size of the array.

When searching for the index of a particular value in a partially-filled array, the array should be declared const.

When inserting an element into a partially-filled array, it is an error if size >= capacity.

In a partially-filled array, the size may be less than the array's capacity.

When comparing two partially-filled arrays for equality, both arrays should be declared const.

When deleting an element from a partially-filled array, it is an error if the index of the element to be removed is >= size.

In a partially-filled array, the size represents the effective size of the array.

When inserting elements into a partially-filled array, the array should not be declared const.

[1701] Where are the characters "Hello" stored in memory?

```
char s1[1024] = "Hello";
void f()
{
const char *s2 = "Goodbye";
char s3[] = "CS 150";
}
```

stack
heap
static storage area (read-only)
static-storage area (read/write)

static-storage area (read/write)

[1702] Where are the characters "Goodbye" stored in memory?

```
char s1[1024] = "Hello";
void f()
{
const char *s2 = "Goodbye";
char s3[] = "CS 150";
}
```

stack
heap
static storage area (read-only)
static-storage area (read/write)

static storage area (read-only)

[1703] Where are the characters "CS 150" stored in memory?

```
char s1[1024] = "Hello";
void f()
{
const char *s2 = "Goodbye";
char s3[] = "CS 150";
}
```

stack
heap
static storage area (read-only)
static-storage area (read/write)

stack

```
char s1[1024] = "Hello";
void f()
{
const char *s2 = "Goodbye";
char s3[] = "CS 150";
}
```

stack
heap
static storage area (read-only)
static-storage area (read/write)

[1705] What happens here

```
void f()
{
char * s = "CS 150";
s[0] = 'X';
cout << s << endl;
}
```

Prints "XS 150"
Most likely crashes when run
Code compiles without warnings
Code fails to compile because "CS 150" is const

Most likely crashes when run

[1706] To process array-style (C) strings in C++, use the header:

<string>
<cstring>
<c-string>
"cstring.h"

<cstring>

[1707] What happens here?

```
char * s = "CS150";
strcpy(s, "CS50");
cout << s << endl;
```

The code will not compile
Code will compile (with warnings), but crash when run

"CS50"
"CS500"
"CS150CS50"

Code will compile (with warnings), but crash when run.

[1708] What happens here?

```
char s[] = "CS150";
strcpy(s, "CS50");
cout << s << endl;
```

Crashes when run
Undefined behavior

"CS50"
"CS500"
"CS150CS50"

"CS50"

[1709] What happens here?

```
char s[] = "CS150";
strcat(s, "CS50");
cout << s << endl;
```

Crashes when run
Undefined behavior

"CS50"
"CS500"
"CS150CS50"

Undefined behavior

char s[50] = "CS150";
strcat(s, "CS50");
cout << s << endl;

Crashes when run
Undefined behavior

"CS50"
"CS500"
"CS150CS50"

[1711] What happens here?

char s1[] = "CS150";
char *s2 = s1;
s2[0] = 'X';
cout << s1 << endl;

"XS150"
"CS150"

Crashes when run
Does not compile
Undefined behavior

"XS150"

[1712] What happens here?

char *s1 = "CS150";
char s2[] = s1; // C++ forbids converting a string constant to 'char*'

s2[0] = 'X';
cout << s1 << endl;

"XS150"
"CS150"

Crashes when run
Does not compile
Undefined behavior

Does not compile

[1713] What happens here?

char s1[] = "CS150", s2[10];
strcpy(s2, s1);
s2[0] = 'X';
cout << s1 << endl;

"XS150"
"CS150"
Does not compile
Crashes when run.
Undefined behavior

"CS150"

[1714] What happens here?

char s1[] = "CS150", s2[10];
strcpy(s1, s2);
s2[0] = 'X';
cout << s1 << endl;

"XS150"
"CS150"
Does not compile
Crashes when run.
Undefined behavior

Undefined behavior

[1715] What is true about a?

char a[] = "Sup?";

It is an array with sizeof 4
It is an array with sizeof 5
It is a C-string with strlen 5
It is a pointer to an array of 4 characters

It is an array with sizeof 5

const char **a = "dog",** b = a;
if (strcmp(a, b)) cout << "dog == dog" << endl;
else cout << "dog != dog" << endl;

dog != dog
dog == dog
Crashes when run
Does not compile

[1717] What prints here?

const char **a = "dog",** b = a;
if (a == b) cout << "dog == dog" << endl;
else cout << "dog != dog" << endl;

dog != dog
dog == dog
Crashes when run
Does not compile

dog == dog

[1718] What is the result of running this line of code?

char s[] = "hi\0hey";

3 chars 'h', 'i', '\0' stored in s. strlen(s) is 2.
6 chars, 'h','i','\0','h','e','y' stored in s. strlen(s) is 2.
7 chars, 'h','i','\0','h','e','y','\0' stored in s. strlen(s) is 2.
7 chars, 'h','i','\0','h','e','y','\0' stored in s. strlen(s) is 6.
This is a syntax error.

7 chars, 'h','i','\0','h','e','y','\0' stored in s. strlen(s) is 2.

[1719] Which of these is a legal assignment?

string name = "Houdini";

string str = c_str(name);
char* cstr = name.c_str();
string* strp = name.c_str();
const char *cstr = c_str(name);
const char *cstr = name.c_str();

const char *cstr = name.c_str();

[1720] Which line makes the comment correct?

char s[50];
char *t = "ac";
// Make s into a C-string "ac"

s = t;
s = "ac";
s[0] = t[0]; s[1] = t[1]; s[2] = t[2];
None of these
s[0] = t[0]; s[1] = t[1];

s[0] = t[0]; s[1] = t[1]; s[2] = t[2];

[1721] Which lines create the C-string "hello"?

1. char s[10] = "hello";
2. char s[10] = {'h','e','l','l','o'};
3. char s[] = {'h','e','l','l','o','0'};
4. char s[5] = "hello";
5. char s[] = "hello";

1, 2, 3, 5
1, 2, 5
All of them
1, 3
1, 5

1, 2, 5

[1722] Which lines contains exactly two characters?

1. "\n"
2. '\n'
3. "n"
4. "/n"
5. 'n'

1, 3, 5
1, 2, 4
All of them
1, 3, 4
1, 3

1, 3

void stringCopy(char ***p, const char** *q)
{
while ((***p =** *q) != '\0') {
p++;
q++;
}
}
 	
No, because there is no *p = '\0'; after the loop
No, because the comparison should be against 0, not against '\0'
No, because the condition accidentally used = instead of ==
Yes, the terminator is copied as the condition fails
No, because there is no actual copy of characters into p at all	

[1724] Which while condition makes this function correct?
 	
int stringComp(const char ***s1, const char** * s2)
{
while (. . .) { s1++; s2++; }
return ***s1 -** *s2
}
 	
***s1 !=** *s2
***s1 ==** *s2
***s1 &&** *s2
***s1 ==** ***s2 ||** ***s1 ||** *s2
***s1 ==** ***s2 &&** ***s1 &&** *s2
	***s1 ==** ***s2 &&** ***s1 &&** *s2

[1725] Which library function performs an equivalent operation on C-strings? string s1 = "Hello"; string s2 = "World"; s1 = s1 + s2; strlen() strcpy() strcmp() strcat() None of these	strcat()
[1726] Which library function performs an equivalent operation on C-strings? string s1 = "Hello"; string s2 = "World"; s1 = s2; strlen() strcpy() strcmp() strcat() None of these	strcpy()

[1727] Which library function performs an equivalent operation on C-strings?
 	
string s1 = f(), s2 = f();
if (s1 < s2) . . .
 	
strlen()
strcpy()
strcmp()
strcat()
None of these
	strcmp()

[1728] Which library function performs an equivalent operation on C-strings?
 	
string s = mystery();
if (s.size() > 3) . . .
 	
strlen()
strcpy()
strcmp()
strcat()
None of these
	strlen()

The characters for the C-string char * s1 = "hello"; are stored in user memory and may be modified.

strcmp(s1, s2) returns true if s1 and s2 contain the same characters.

The strlen() function returns the allocated size of a C-string allocated as an array.

The C-string type is part of the standard library, not built into the C++ language.

C-string assignment uses the = operator.

The length of a C-string is stored explicitly in its length data member

The allocated size for the C-string char s1[1024] = "hello"; is 6 characters, while the effective size is 5 characters.

C-string assignment uses the strcat() function.

The strcat() function cannot overflow the storage allocated for the destination buffer.

The strncpy() function always appends a trailing NUL when the copy is finished.

strcmp(s1, s2) returns a negative number if s1 is lexicographically "greater than" s2.

The sizeof operator returns the effective size of a C-string allocated as an array.

The strncpy() function is straightforward and easy to use.

strcmp(s1, s2) returns a positive number if s1 is lexicographically "less than" s2.

You can compare two C-strings, s1 and s2, by using the == operator.

C-strings use the + operator for concatenation.

C-strings are char pointers to the first character in a sequence of characters, terminated with a '0' character.

When writing programs that interact with your operating system, either Windows, Mac OSX or Linux, you will normally use the C++ library string type, rather than the older C-string type.

The C-string literal "cat" contains 3 characters.

The strcpy() function expands the destination string to make sure it is large enough to hold the source string.

You can compare two C-strings, s1 and s2, by using the strcmp() function. C-strings are character arrays that rely on a special embedded sentinel value, the character with the ASCII code 0. The allocated size for the C-string char s1[] = "hello"; is 6 characters, while the effective size is 5 characters. The sizeof operator returns the allocated size of a C-string allocated as an array. The effective size of the C-string char * s1 = "hello"; is 5 characters, but 6 characters are used for storage. strcmp(s1, s2) returns a positive number if s1 is lexicographically "greater than" s2. C-strings use the strcat() function for concatenation. The strlen() function returns the effective size of a C-string. C-strings are often needed to interoperate with legacy C libraries. When writing programs that interact with your operating system facilities, either Windows, Mac OSX or Linux, you will normally use C-strings instead of the C++ library string type. The characters for the C-string char s1[] = "hello"; are stored in user memory and may be modified. C-strings are char pointers to the first character in a sequence of characters, terminated with a '\0' character. C-string functions may be more efficient than C++ string member functions. strcmp(s1, s2) returns a negative number if s1 is lexicographically "less than" s2. Given the C-string char * s3 = "hello"; strlen(s3) returns 5. C-string assignment uses the strcpy() function. strcmp(s1, s2) returns 0 if s1 and s2 contain the same characters. The C-string type is built into the C++ language, not defined in the standard library. The strcpy() function always appends a trailing NUL when the copy is finished. The strncpy() function can be used to make sure that you don't copy more characters than necessary. Programs written for embedded devices often use C-strings rather than the C++ library string type. The length of a C-string is never stored explicitly The C-string literal "cat" contains 4 characters. The strncat() function allows you to limit the maximum number of characters that are concatenated. The character with the ASCII code 0 is called the NUL character	
[1801] Which of these is a 2D array? int d[][] int *b[2]; int a[][2]; int c[2][2];	int c[2][2];

[1802] Which function prototype could process a 2D array?
 	
void f(int **a);
void f(int[][] a);
void v(int a[][]);
void f(int a[2][]);
void (f(int a[][2]);
	void (f(int a[][2]);

[1803] What prints? Assume 4 bytes per int. int a[][2] = {0}; cout << sizeof(a) << endl; 16 12 4 8 Illegal declaration. Does not compile.	8

int a[][2] = {{0},{0}};
cout << sizeof(a) << endl;

4
12
16
8

Illegal declaration. Does not compile.

[1805] What prints? Assume 4 bytes per int.

int a[][2] = {1, 2, 3};
cout << sizeof(a) << endl;

8
12
4
16

Illegal declaration. Does not compile.

16

[1806] What prints? Assume 4 bytes per int.

int a[][] = {{1, 2}, {3, 4}};
cout << sizeof(a) << endl;

4
12
16
8

Illegal declaration. Does not compile.

Illegal declaration. Does not compile.

[1807] What prints?

int a[4][2] = {1, 2, 3, 4, 5, 6, 7};
cout << a[2][1] << endl;

Undefined (out of bounds)
6
Illegal declaration. Does not compile.
5
4

// 0 1
// 0 { 1, 2
// 1 3, 4
// 2 5, 6
// 3 7, 0 }

6

[1808] Which one of the following statements is the correct definition for a two-dimensional array of 20 rows and 2 columns of the type integer?

int num[2, 20]
int num[2][2];
int num[20][2];
None of these
int num[20, 2];

int num[20][2];

[1809] Which statement displays the value 24 from the 2D array initialized here?

int a[2][3] =
{
{ 13, 23, 33 },
{ 14, 24, 34 }
};

cout << a[2][2];
cout << a[1][2];
cout << a[1][1];
cout << a[2][1];
None of these

cout << a[1][1];

[1810] Which value of a is stored in the val variable?

auto val = a[0][2];
The value in the first row and the third column
The value in the third row and the first column
The value in the first row and the second column
None of these
The value in the first row and the first column

The value in the first row and the third column

cout << a[3][2];
cout << a[2][1];
None of these
cout << a[2][3];
cout << a[1][2];

[1812] What prints when this runs?

```
int a[2][3] = {1, 2, 3, 4, 5, 6};
cout << a[0][2] + a[1][2] << endl;
```

5
10
7
8
9

// 0, 1, 2
// 0 { 1, 2, 3
// 1 4, 5, 6 }

9

[1813] What is the value of a[1][1] after this runs?

```
int cnt = 0, a[2][3];
for (int i = 0; i < 3; i++)
for (int j = 0; j < 2; j++)
a[j][i] = ++cnt;
```

6
4
2
3
5

4

[1814] What is the value of a[1][2] after this runs?

```
int cnt = 0, a[2][3];
for (int i = 0; i < 3; i++)
for (int j = 0; j < 2; j++)
a[j][i] = ++cnt;
```

4
6
2
5
3

6

[1815] What is the value of a[0][2] after this runs?

```
int cnt = 0, a[2][3];
for (int i = 0; i < 3; i++)
for (int j = 0; j < 2; j++)
a[j][i] = ++cnt;
```

6
4
2
5
3

5

[1816] What prints?

```
int a[2][3] = {{3,2,3}};
cout << a[0][0] << a[1][0] << endl;
```

00
Code does not compile
31
30
33

30

[1817] What prints?

```
int a[3][2] = {{3,2,3}}; // too many initializers for 'int [2]'
cout << a[0][0] << a[1][0] << endl;
```

00
30
Code does not compile
33
31

Code does not compile

int a[3][2] = {3,2,3};
cout << a[0][0] << a[1][0] << endl;

Code does not compile
31
33
00
30

[1819] What prints?

int cnt = 0, a[4][5];
for (int i = 0; i < 5; i++)
for (int j = 0; j < 4; j++)
a[j][i] = cnt++;
cout << a[1][2] << endl;

11
9
19
8
14

9

[1820] What prints?

int cnt = 0, a[4][5];
for (int i = 0; i < 5; i++)
for (int j = 0; j < 4; j++)
a[j][i] = cnt++;
cout << a[2][3] << endl;

19
14
11
9
8

14

[1821] What prints?

int cnt = 0, a[4][5];
for (int i = 0; i < 5; i++)
for (int j = 0; j < 4; j++)
a[j][i] = cnt++;
cout << a[3][2] << endl;

14
11
9
8
19

11

[1822] What prints?

int cnt = 0, a[4][5];
for (int i = 0; i < 5; i++)
for (int j = 0; j < 4; j++)
a[j][i] = cnt++;
cout << a[3][2] << endl;

11
8
9
14

11

[1823] How many rows are in this array?

int a[2][3];

6
3
5
4
2

2

[1824] How many columns are in this array?

int a[2][3];

3
6
4
5
2

3

int a[2][3];

3
2
5
4
6

[1826] How many (int[]) elements are in this array?

int a[2][3];

5
3
2
4
6

2

[1827] What is the correct version of main() if you wish to process command-line arguments?

int main(int argc, char* argv[])
int main(int argc[], char * argv)
int main(char *argc, int argv[])
int main(char argv[], int argc)
int main(string args[])

int main(int argc, char* argv[])

[1828] What is true about the command line in C++?

I. The first argument is the name of the program
II. Command line arguments are passed in an array
III. Use main(int argc, char* argv[])

I and III
All of these are true
I only
II and III
II only

All of these are true

[1829] Why is the command-line argc always at least 1?

Because argv[0] is the name of the program running
Because the argv[] array is a special case that starts at 1
Because argv[0] is unused
Because argv[0] is a pointer named this
It is not. If there are no arguments passed, then argc is 0

Because argv[0] is the name of the program running

[1830] The program a.out is run like this:

./a.out alex brent chris rodger 32 33 44 78

argc is 9 and argv[0] is "./a.out"
argc is 8 and argv[0] is "alex"
argc is 9 and argv[0] is "alex"
argc is 9 and argv[0] is "./a"
argc is 8 and argv[0] is "./a.out"

argc is 9 and argv[0] is "./a.out"

[1831] What prints?

```
int a[5][3] = {
{ 1, 2, 3},
{ 4, 5, 6},
{ 7, 8, 9},
{10, 11, 12},
{13, 14, 15}
};

int *p = &a[0][0];
cout << p[1][2] << endl; // invalid types 'int[int]' for array subscript
```

Undefined (out of bounds)
6
2
Illegal; will not compile
An address

Illegal; will not compile

int a[5][3] = {
{ 1, 2, 3},
{ 4, 5, 6},
{ 7, 8, 9},
{10, 11, 12},
{13, 14, 15}
};

int *p = &a[0][0];
cout << *p << endl;

4
Illegal; will not compile
1
An address
Undefined (out of bounds)

[1833] What prints?

int a[5][3] = {
{ 1, 2, 3},
{ 4, 5, 6},
{ 7, 8, 9},
{10, 11, 12},
{13, 14, 15}
};

int *p = &a[0][0];
cout << (p + 5) << endl;

4
An address
Illegal; will not compile
1
Undefined (out of bounds)

An address

[1834] What prints?

int a[5][3] = {
{ 1, 2, 3},
{ 4, 5, 6},
{ 7, 8, 9},
{10, 11, 12},
{13, 14, 15}
};

int *p = &a[0][0];
cout << p[10] << endl;

Illegal; will not compile
An address
10
11
Undefined (out of bounds)

11

[1835] What prints?

int a[5][3] = {
{ 1, 2, 3},
{ 4, 5, 6},
{ 7, 8, 9},
{10, 11, 12},
{13, 14, 15}
};

int *p = &a[0][0];
cout << (p + 5 * 2)[1] << endl;

13
12
11
Undefined (out of bounds)
Illegal; will not compile

12

int x = 0;
int a[2][3] = {{1, 2, 3}, {4, 5, 6}};
for (auto r : a)
for (auto c : r) x++; // 'r' was not declared in this scope
cout << x << endl;
 	
Undefined (out of bounds)
6
Illegal; will not compile
2
3	

[1837] What prints?
 	
int x = 0;
int a[2][3] = {{1, 2, 3}, {4, 5, 6}};
for (const auto& r : a)
for (const auto& c : r) x++;
cout << x << endl;
 	
3
2
Undefined (out of bounds)
6
Illegal; will not compile
	6

[1838] What prints?
 	
int x = 0;
int a[2][3] = {{1, 2, 3}, {4, 5, 6}};
for (const auto& r : a)
for (const auto& c : r) x += c;
cout << x << endl;
 	
21
Undefined (out of bounds)
15
6
Illegal; will not compile
	21

[1839] What prints?
 	
int x = 0;
int a[2][3] = {{1, 2, 3}, {4, 5, 6}};
for (const auto& r : a)
for (const auto& c : r) x = c;
cout << x << endl;
 	
21
Undefined (out of bounds)
6
15
Illegal; will not compile
	6

You can pass the first row of the 2D array int a[3][3] to the function f(int *a, size_t n) by calling f(a[0], 3).	True
You can pass the 2D array int a[3][3] to the function f(int *a, size_t r, size_t c) by calling f(&a[0][0], 3, 3).	True
Physically, a 2D array is stored as a single linear, contiguous array with the elements for each column following the elements for the previous column in memory.	False
Command line arguments that start with a hyphen are usually called switches.	True
You can use a range-based loop on a 2D array.	True
You can pass the 2D array int a[3][3] to the function f(int a[3][], size_t n) by calling f(a, 3).	False
Physically, a 2D array is stored as a single linear, contiguous array with the elements for each row following the elements for the previous row in memory.	True
A 2D array address expression is the equivalent of: **(address + (row** height + col))	False
You cannot use a range-based loop on a 2D array.	False
You can pass the first column of the 2D array int a[3][3] to the function f(int *a, size_t n) by calling f(a[0], 3).	False
You can pass the 2D array int a[3][3] to the function f(int a[][3], size_t n) by calling f(a, 3).	True
In a 2D array the first subscript represents the rows and the second the columns.	True

A 2D array address expression is the equivalent of: **(address + (row** width + col))	True
When initializing a 2D, each row must have its own set of braces.	False
When passing a 2D array to a function, the array parameter must explicitly list the size for all dimensions except for the last, like: void f(int a[3][], size_t n);	False
A 2D array is a 1D array whose elements are also 1D arrays.	True
The rules for partial initialization of a 2D array can be changed by adding braces around interior array elements.	True
You can pass the 2D array int a[3][3] to the function f(int a[][], size_t r, size_t c) by calling f(a, 3, 3).	False
Your operating system's command processor is known as the shell.	True
When initializing a 2D, each column must have its own set of braces.	False
You can pass the 2D array int a[3][3] to the function f(int *a, size_t r, size_t c) by calling f(a, 3, 3).	False
Conceptually, a 2D array is rectangular grid of columns and rows.	True
Physically, a 2D array is stored as a rectangular grid of columns and rows.	False
When passing a 2D array to a function, the array parameter must explicitly list the size for all dimensions except for the first, like: void f(int a[][3], size_t n);	True
In a 2D array the first subscript represents the columns and the second the rows.	False
On the command line, argc is the count of arguments including the program itself.	True

[1901] The variable p is located:
 	
void f()
{
int *p = new int;
}
 	
in the static storage area
None of these
on the heap
on the stack
	on the stack

[1902] The variable *p is located:
 	
void f()
{
int *p = new int;
}
 	
on the heap
None of these
in the static storage area
on the stack
	on the heap

[1903] The variable *p:
 	
void f( )
{
int *p = new int;
}
 	
Stores a memory address
Stores the value 0
It's uninitialized
	It's uninitialized

[1904] The variable p:
 	
void f( )
{
int *p = new int;
}
 	
stores the value 0
stores a memory address
None of these
is uninitialized
	stores a memory address

void f( )
{
int *p = new int{42};
}

It's undefined → Code does not compile

Stores a memory address

Stores the value 42 in all versions of C++

Stores the value 42 in C++11 only

It's uninitialized

[1906] The variable *p:

void f( )
{
int *p = new int(42);
}

It's undefined → Code does not compile.
Stores the value 42 in all versions of C++
Stores a memory address
It's uninitialized
Stores the value 42 in C++11 only

Stores the value 42 in all versions of C++

[1907] The variable *p:

void f( )
{
int *p = new int{};
}

Stores a memory address
It's uninitialized
Stores the value 0 in all versions of C++
Stores the value 0 in C++11 only
It's undefined → Code does not compile.

Stores the value 0 in C++11 only

[1908] The variable *p:

void f( )
{
int *p = new int = {42};
}

Stores the value 42 in all versions of C++
It's undefined → Code does not compile.
It's uninitialized
Stores a memory address
Stores the value 42 in C++11 only

It's undefined → Code does not compile.

[1909] The variable *p:

void f( )
{
string *p = new string;
}

It's undefined → Code does not compile
Stores an empty string
Stores nullptr
It's uninitialized
Stores a memory address

Stores an empty string

void f( )
{
int *p = new int[42];
}

It's undefined Code → does not compile
The first element of an array of 42 uninitialized ints
A single int with the value 42
The first element of an array of 42 ints with the value 0

[1911] The variable p points to:

void f( )
{
int *p = new int[42]();
}

The first element of an array of 42 uninitialized ints
The first element of an array of 42 ints with the value 0
A single int with the value 42
It's undefined → Code does not compile

The first element of an array of 42 ints with the value 0

[1912] The variable p points to:

void f()
{
int *p = new int[3]{1, 2, 3};
}

---

is undefined. Code does not compile.
the first element of an array of 3 uninitialized ints
a single int with the value 1
the first element of an array of 3 ints with the values 1,2,3

---

the first element of an array of 3 ints with the values 1,2,3

[1913] The variable p points to:

void f()
{
int *p = new int[3] = {1, 2, 3};
}

the first element of an array of 3 ints with the values 1,2,3
the first element of an array of 3 uninitialized ints
a single int with the value 1
is undefined. Code does not compile

is undefined. Code does not compile.

[1914] Examine this code. What goes on the blank line?

void f()
{
int *p = new int[3]{1, 2, 3};
. . .
_____
}

delete *p;
delete[] p;
delete p[3];
None of these is correct
delete p;

delete[] p;

[1915] Examine this code. What goes on the blank line?

void f()
{
int *p = new int[3]{1, 2, 3};
. . .
_____
}

delete p[];
delete p;
delete p[3];
None of these is correct
delete[] *p;

None of these is correct

void f()
{
int *p = new int[3]{1, 2, 3};
. . .
_____
}

delete[] p;
delete *p;
None of these is correct
delete p[0]; delete[p1]; delete [p2];
delete p[];

[1917] Examine this code. What goes on the blank line?

void f()
{
int *p = new int{3};
. . .
_____
}

delete p[3];
delete[] p;
None of these is correct
delete p;
delete *p;

delete p;

[1918] Examine this code. What goes on the blank line?

void f()
{
int *p = new int{3};
. . .
_____
}

delete *p;
delete p[];
None of these is correct
delete[] p;
delete p[3];

None of these is correct

[1919] This code:

void f( )
{
int *p = new int[3]{rand(), rand(), rand()};
if (p[1] == 0 || p[2] == 0)
throw "Divide by 0";
cout << p[0] / p[1] / p[2] << endl;
delete[] p;
}

has a syntax error
None of these
has a double delete
has a memory leak
has a dangling pointer

has a memory leak

[1920] This code:

void f()
{
int *p = new int[3]{rand(), rand(), rand()};
if (p[1] == 0 || p[2] == 0) return;
cout << p[0] / p[1] / p[2] << endl;
delete[] p;
}

has a memory leak
has a syntax error
has a double delete
None of these
has a dangling pointer

has a memory leak

void f()
{
int *p = new int[3]{rand(), rand(), rand()};
if (p[1] != 0 && p[2] != 0)
cout << p[0] / p[1] / p[2] << endl;
delete[] p;
}
 	
None of these
has a double delete
has a syntax error
has a memory leak
has a dangling pointer	

[1922] This code:
 	
void f()
{
int *p = new int[3]{rand(), rand(), rand()};
if (p[1] != 0 && p[2] != 0) delete[] p;
cout << p[0] / p[1] / p[2] << endl;
}
 	
has a dangling pointer
has a syntax error
has a double delete
None of these
has a memory leak
	has a dangling pointer

[1923] This code:
 	
int * f()
{
int a[] = {1, 2, 3};
return &a[1];
}
 	
None of these
has a dangling pointer
has a syntax error
has a memory leak
has a double delete
	has a dangling pointer

[1924] This code:
 	
void f()
{
int *p = new int[3]{rand(), rand(), rand()};
if (p[1] != 0 && p[2] != 0)
delete[] p;
else
cout << p[0] / p[1] / p[2] << endl;
delete[] p;
}
 	
has a double delete
None of these
has a dangling pointer
has a syntax error
has a memory leak
	has a double delete

[1925] To use any of C++ smart pointer types, include the header: <memory> <ptr> <new> <smart_ptr> <alloc>	<memory>
[1926] Which line correctly creates a smart pointer that points to the variable x? int x = 42; unique_ptr<int[]>(&x); make_shared<int>(x); unique_ptr<int>(&x); None of these shared_ptr<int>(&x);	None of these

int x = 42;

unique_ptr<int[]>(&x);
None of these
shared_ptr<int>(&x);
unique_ptr<int>(&x);
make_shared<int>(x);

[1928] What does this code print?

```
int main()
{
auto p1 = make_shared<int>(42);
auto p2 = p1;
cout << *p1 << endl;
cout << *p2 << endl;
(*p2)++;
cout << *p1 << endl;
}
```

424343
Does not compile (illegal)
424242
Undefined behavior
424243

424243

[1929] Given this declaration, which line below is illegal?

```
auto p1 = make_shared<int>(42);
```

```
cout << *p1 << endl;
(*p1)++;
delete p1;
```

None of these are illegal
auto p2 = p1;

delete p1;

[1930] Given this declaration, which line below is illegal?

```
auto p1 = unique_ptr<int>(new int{42});
```

```
(*p1)++;
p1.release();
```

None of these are illegal
auto p2 = p1;
cout << *p1 << endl;

auto p2 = p1;

[1931] What does this code print?

```
int main()
{
auto p1 =unique_ptr<int>(new int{42});
cout << *p1;
auto p2 = p1.release();
cout << *p2;
(*p2)++;
cout << *p2;
}
```

Undefined behavior
424343
424243
424242
Does not compile (illegal)

424243

[1932] What does this code print?

```
int main()
{
auto p1 =unique_ptr<int>(new int{42});
cout << *p1;
auto p2 = p1;
cout << *p2;
(*p2)++;
cout << *p2;
}
```

Does not compile (illegal)
424242
424243
Undefined behavior
424343

Does not compile (illegal)

int main()
{
auto p1 =unique_ptr<int>(new int{42});
cout << *p1;
auto p2 = p1.release(); 🌺
cout << *p2;
(*p2)++;
cout << *p1; 🌺 Isn't called correctly
}
 	
Does not compile (illegal)
Undefined behavior
424242
424243
424343	

[1934] The member function get() returns the raw pointer that a smart pointer contains. What does this code print?
 	
int main()
{
auto p1 =unique_ptr<int>(new int{42});
cout << *p1;
auto p2 = p1.release(); // Resets to NULL Pointer
cout << *p2;
(*p2)++;
cout << p1.get() << endl; // Returns 0
}
 	
424343
42430
Does not compile (illegal)
Undefined behavior
42420
	42420

A unique_ptr uses a reference count to manage how many pointers point to an object.	False
To allocate memory on the heap, C++ uses the new operator.	True
Memory for global variables is allocated when the program is loaded from disk. This is known as dynamic allocation.	False
The statement new int{3}; allocates an array of three integers on the heap.	False
The statement new int{3}; allocates a single initialized integer on the heap.	True
Memory for local variables is allocated on the stack when their definitions are encountered during runtime. This is known as dynamic allocation.	False
Requesting a block of memory from the operating system as the program runs is known as static allocation.	False
Memory for global variables is allocated when the program is loaded from disk. This is known as static allocation.	True
Smart pointers may point to objects allocated on the stack.	False
Assuming p is a pointer to a single variable allocated on the heap, the statement delete[] p; returns the allocated memory back to the operating system for reuse.	False
A pointer that goes out of scope before deleting the memory it points to is called a double delete.	False
A pointer-like object that can be used to automatically manage memory allocated on the heap is called a smart pointer.	True
Memory for global variables is allocated when the program is loaded from disk. This is known as automatic allocation.	False
The release() function returns the raw pointer that a unique_ptr contains before seting the pointer to nullptr.	True
Assuming p is a pointer to a single variable allocated on the heap, the statement delete p; sets the pointer to nullptr so that the memory can be reused for another allocation.	False
Memory for local variables is allocated on the stack when their definitions are encountered during runtime. This is known as static allocation.	False
The statement new int{3}; allocates an array of three integers on the heap	False
Using a pointer to access the memory it points to after the pointer has been deleted is called a double delete.	False

A pointer that goes out of scope before deleting the memory it points to is called a memory leak.	True
If the new operator cannot allocate memory, C++ throws an exception.	True
The statement new int{}; is a syntax error.	False
Using a pointer to access the memory it points to after the pointer has been deleted is called a memory leak.	False
The reset() function returns the raw pointer that a unique_ptr contains, before setting that pointer to nullptr.	False
To transfer a unique_ptr to a vector, use push_back along with the move() function.	True
The statement new int[3]{1, 2, 3}; allocates an array of three initialized integers on the heap.	True
Requesting a block of memory from the operating system as the program runs is known as dynamic allocation.	True
Assuming p is a pointer to the first variable in an array allocated on the heap, the statement delete[] p; returns the allocated memory back to the operating system for reuse.	True
The statement new int[3] = {1, 2, 3}; is a syntax error.	True
The statement new int[3](); allocates an array of three default-initialized integers on the heap.	True
The statement new int[3]; allocates an array of three uninitialized integers on the heap.	True
The statement new int{}; allocates a default-initialized integer on the heap.	True
Freeing unused memory that was allocated elsewhere in your program is done in C++ using a garbage collector.	False
The statement new int[3](); is a syntax error.	False
Using a pointer to access the memory it points to after the pointer has been deleted is called a dangling pointer.	True
A unique_ptr can refer to a dynamic array.	True
A unique_ptr may transfer its ownership to another unique_ptr.	True
Assuming p is a pointer to a single variable allocated on the stack, the statement delete p; returns the allocated memory back to the operating system for reuse.	False
Freeing unused memory that was allocated elsewhere in your program is done in C++ using manual memory management.	True
The release() function deletes the raw pointer that a unique_ptr contains, and then sets that pointer to a new value.	False
Assuming p is a pointer to the first variable in an array allocated on the heap, the statement delete p; returns the allocated memory back to the operating system for reuse.	False
A shared_ptr uses a reference count to manage how many pointers point to an object.	True
A pointer that goes out of scope before deleting the memory it points to is called a dangling pointer.	False
To allocate memory on the stack, C++ uses the new operator.	False
The statement new int; allocates an uninitialized integer on the heap.	True
Smart pointers automatically delete the memory they point to at the appropriate time.	True
Assuming p is a pointer to a single variable allocated on the heap, the statement delete p; returns the allocated memory back to the operating system for reuse.	True
If the new operator cannot allocate memory, C++ returns nullptr.	False
The statement new int; allocates an uninitialized integer on the stack.	False
The statement new int[3] = {1, 2, 3}; allocates an array of three initialized integers on the heap.	False

A pointer-like object that can be used to automatically manage memory allocated on the heap is called a raw pointer.	False
Requesting a block of memory from the operating system as the program runs is known as automatic allocation.	False

[2001] Which item is a mutator?
 	
class Alligator
{
public:
Alligator(double w);
void eat();
string toString() const;
private:
double weight;
};
 	
None of these
toString()
weight
eat()
Alligator()
	eat()

[2002] Which of these is a default constructor?
 	
class Alligator
{
public:
Alligator(double w);
void eat();
string toString() const;
private:
double weight;
};
 	
None of these
Alligator()
toString()
weight
eat()
	None of these

[2003] Which of these is a constructor?
 	
class Alligator
{
public:
Alligator(double w);
void eat();
string toString() const;
private:
double weight;
};
 	
weight
toString()
Alligator()
None of these
eat()
	Alligator()

[2004] Which of these is an accessor?
 	
class Alligator
{
public:
Alligator(double w);
void eat();
string toString() const;
private:
double weight;
};
 	
toString()
weight
Alligator()
None of these
eat()
	toString()

class Alligator
{
public:
Alligator(double w);
void eat();
string toString() const;
private:
double weight;
};
 	
toString()
Alligator()
eat()
weight
All of these	

[2006] What type of member function is eat()?
 	
class Alligator
{
public:
Alligator(double w);
void eat();
string toString() const;
private:
double weight;
};
 	
mutator
None of these
destructor
accessor
constructor
	mutator

[2007] What type of member function is toString()?
 	
class Alligator
{
public:
Alligator(double w);
void eat();
string toString() const;
private:
double weight;
};
 	
constructor
None of these
mutator
accessor
destructor
	accessor

[2008] What is true about user-defined types implemented using structures with public data members? you cannot enforce the invariant properties of your types modifications to the character of the data members requires clients to rewrite code they may be more error-prone than types developed using classes clients can directly modify the data members of a variable	ANSWER → All of these
[2009] What is true about user-defined types implemented using classes with private data members? clients can directly modify the data members of a variable it is not possible to create immutable objects All of these you can enforce the invariant properties of your types modifications to the character of the data members requires clients to rewrite code	you can enforce the invariant properties of your types
[2010] What is true about user-defined types implemented using classes with private data members? clients can directly modify the data members of a variable you cannot enforce the invariant properties of your types it is not possible to create immutable objects All of these modifications to the character of the data members does not require clients to rewrite code	modifications to the character of the data members does not require clients to rewrite code

it is possible to create immutable objects All of these modifications to the character of the data members requires clients to rewrite code clients can directly modify the data members of a variable you cannot enforce the invariant properties of your types	
[2012] The _____ of a class specifies how clients interact with a class. public interface private implementation private interface public implementation None of these	public interface

[2013] What is f()?
 	
class X
{
public:
X(int);
void f() const;
int g() const;
void h(int);
};
 	
None of these
mutator
destructor
constructor
accessor
	None of these

[2014] What is g()?
 	
class X
{
public:
X(int);
void f() const;
int g() const;
void h(int);
};
 	
None of these
mutator
accessor
destructor
constructor
	accessor

[2015] What is h()?
 	
class X
{
public:
X(int);
void f() const;
int g() const;
void h(int);
};
 	
mutator
destructor
constructor
accessor
None of these
	mutator

[2016] What is X()?
 	
class X
{
public:
X(int);
void f() const;
int g() const;
void h(int);
};
 	
accessor
destructor
mutator
constructor
None of these
	constructor

class Val
{
int data_;
public:
Val(int);
int get() const;
void print() const;
};

void Val::get() { return data_; }
Val::Val(int n) { data_ = n; }
void Val::print() const { cout << data_; }
None of these
Val()
print()
data_
get()

[2018] Which element is private?

struct Val
{
int data_;
public:
Val(int);
int get() const;
void print() const;
};

void Val::get() { return data_; }
Val::Val(int n) { data_ = n; }
void Val::print() const { cout << data_; }

Val()
None of these
get()
print()
data_

None of these

[2019] What is true about a mutator member function?

None of these
It changes one or more data members
It return information about an object's internal state
Its prototype ends with const
Its presence means that a class is immutable

It changes one or more data members

[2020] What is true about an accessor member function?

It is never used when a class is immutable
It returns information about an object's state
It changes one or more data members
Its prototype may not include the keyword const
None of these

It returns information about an object's state

[2021] What is true about the implementation?

class Time {
public:
Time();
long get() const;
void set(long);
private:
long seconds;
};

The accessor and the mutator
The constructor
The data member seconds
All of these are part of the implementation
None of these are part of the implementation

The data member seconds

class Time {
Time();
long get() const;
void set(long);
private:
long seconds;
};

The accessor and the mutator
All of these are part of the implementation
None of these are part of the implementation
The data member seconds
The constructor

[2023] What is the semantic error in this class definition?

class Time {
long seconds;
public:
Time();
long get() const;
void set(long);
};

get() should not have const at the end
seconds should be in the private section
get() is missing an argument
There is no semantic error.
set() is missing const at the end

There is no semantic error.

[2024] What is the semantic error in this class definition?

class Time {
long seconds;
public:
Time();
long get();
void set(long);
};

There is no semantic error.
seconds should be in the private section
get() is missing an argument
set() is missing const at the end
get() is missing const at the end

get() is missing const at the end

[2025] What is the semantic error in this class definition?

class Time {
long seconds;
public:
Time();
long get() const;
void set(long) const;
};

seconds should be in the private section
get() is missing an argument
set() should not have const at the end
There is no semantic error.
get() should not have const at the end

set() should not have const at the end

[2026] What prints here?

```
class Car {
double speed;
public:
Car();
Car(double s);
double get() const;
};
Car::Car() { speed = 10; }
Car::Car(double s) { speed = s; }
double Car::get() const { return speed; }

int main()
{
Car c1, c2(5);
cout << c1.get() << c2.get() << endl;
}
```

Does not compile; c1 is not an object
15
Undefined; c1 not initialized
05
105

105

```
class Car {
double speed;
public:
Car();
Car(double s);
double get() const;
};
Car::Car() { speed = 10; }
Car::Car(double s) { speed = s; }
double Car::get() const { return speed; }

int main()
{
Car c1(), c2(5);
cout << c1.get() << c2.get() << endl;
}
```

Undefined; c1 not initialized
05
15
105
Does not compile; c1 is not an object

[2028] A user-defined type created as a struct

encapsulates its data to prevent accidental modification
can be easily modified without affecting code that uses it.
has its implementation as its interface
is an interface paired with an implementation
enforces type invariants

has its implementation as its interface

[2029] A Fraction denominator must not ever become 0. You can enforce this invariant through:

```
class Fraction {
. . .
public:
Fraction(int, int);
Fraction get() const;
Fraction set(int, int);
};
```

the implementation of the accessor member
the selection of data members
the implementation of the mutator member
by using the access modifier private in place of public
the implementation of a destructor

the implementation of the mutator member

[2030] On the second line of this code, the object named myRadio is:

```
Radio myRadio(98.6, 8);
cout << myRadio.frequency() << endl;
```

an implicit parameter
an instance variable
a function modifier
an explicit parameter
the function return value

an implicit parameter

[2031] The attributes of this class are model and price. In C++ terminology, these are called:

```
class Mobile {
std::string model;
double price;
public:
. . .
};
```

data members
instance variables
class variables
data attributes
fields

data members

class Radio {
public:
Radio();
explicit Radio(double);
Radio(double, int);

double frequency() const;
double frequency(double);
};

constructor
mutator
data member
accessor
method

[2033] In C++ terminology, frequency() is called a:

class Radio {
public:
Radio();
explicit Radio(double);
Radio(double, int);

double frequency() const;
double frequency(double);
};

constructor
method
accessor
data member
mutator

accessor

[2034] In C++ terminology, the two members named frequency() are:

class Radio {
public:
Radio();
explicit Radio(double);
Radio(double, int);

double frequency() const;
double frequency(double);
};

non-member functions
methods
constructors
data members
member functions

member functions

[2035] The default constructor is:

class Radio {
public:
Radio();
explicit Radio(double);
Radio(double, int);

double frequency() const;
double frequency(double);
};
None of these
Radio()
Radio(double, int)
Radio(double)

Radio()

[2036] There is no constructor for this class.

class Integer {
int value_ = 0;
public:
int get() const;
int set(int n);
};

You can create objects; value_ is initialized to 0
The code compiles, but you cannot create objects from this class
You can create objects; value_ is uninitialized
The code will not compile without a constructor

You can create objects; value_ is initialized to 0

class Integer {
int value_;
public:
int get() const;
int set(int n);
};
 	
You can create objects; value_ is initialized to 0
The code compiles, but you cannot create objects from this class
You can create objects; value_ is uninitialized
The code will not compile without a constructor	

With classes, the public interface includes the member functions that allow clients to access object data in a safe way as well as the data members themselves.	False
In C++ you use the keyword public or private to create a section that indicates the access privileges of subsequent data members or member functions.	True
Member functions that change the state of an object are called constructors.	False
The member function int hours() const; provides read-write access to the hours property (however it is stored).	False
A class is an interface paired with an implementation.	True
Mutator member functions are allowed to read data members, but not change them.	False
Member functions that change the state of an object are called accessors.	False
Accessor member functions should always end in the keyword const.	True
If your class does not have a constructor, the compiler will synthesize a working constructor for you.	False
If a member function is in the private section of a class, it cannot be called from other member functions of the class.	False
The implementation of a class normally appears entirely inside the class .cpp file.	False
The implementation of a member function from the Time class would look something like this: int Time.hours() const {...}.	False
The interface of a class includes all items in the header file.	False
The interface of a class includes all public items in the header file.	True
When calling a member function, like t.hours(3); the address of the object t is passed to the function implicitly as the first parameter.	True
Member functions that initialize the data members of a new object are called mutators.	False
If you write a working constructor for your class, C++ will remove the synthesized default constructor.	True
The implementation of a member function from the Time class would look something like this: int Time::hours() const {...}	True
With classes, the public interface includes the member functions that allow clients to access object data in a safe way.	True
In C++ there is actually no difference between structures and classes.	False
A class definition ends with a semicolon.	True
The implementation of a class includes all private data members in the header file.	True
In C++ you use the keyword public or private before each data member or member function to indicate its access privileges.	False
In C++ there is actually no difference between structures and classes	False
Mutator member functions are allowed to read data members, but not change them	False
Programmers using class-derived objects, directly manipulate the data members of those objects.	False
Using structures for user-defined types means that you cannot change the data representation without affecting the users of your data type.	True
Using classes for user-defined types means that you cannot enforce restrictions on data member access.	False

Programmers using structure-derived variables, directly manipulate the data members of those variables.	True
You may add = default; to the prototype for a default constructor to retain the synthesized version in the presence of other overloaded constructors.	True
The implementation of a class normally appears partly inside the class .cpp file and partly inside the class .h file.	True
If your class does not have a constructor, the compiler will synthesize a default constructor for you.	True
If a member function is in the private section of a class, it can only be called by other member functions of the class.	True
The implementation of a member function from the Time class would look something like this: int hours() const {...}.	False
The two parts of a class are a private interface and a public implementation.	False
The public interface of a class consists of the prototypes of its member functions.	True
Mutator member functions are allowed to read data members and also change them.	True
A structure is an interface paired with an implementation.	False
Using structures for user-defined types means that you can enforce restrictions on data member access.	False
A class definition normally appears in a .h file.	True
In C++ the only difference between structures and classes is that member functions are public by default in structures.	True
Member functions that initialize the data members of a new object are called constructors.	True
Using structures for user-defined types means that you can change the data representation without affecting the users of your data type.	False
Accessor member functions are allowed to read data members, but not change them.	True
The two parts of a class are a public interface and a private implementation.	True
The member function int hours() const; provides read-only access to the hours property (however it is stored).	True
In C++ the only difference between structures and classes is that member functions are private by default in classes.	True
A constructor always has the same name as the class, and no return type.	True
The member function int& hours(); provides read-write access to the hours property (however it is stored).	True
Using structures for user-defined types means that you cannot enforce restrictions on data member access.	True
Using classes for user-defined types means that you can change the data representation without affecting the users of your class.	True
In C++ the only difference between structures and classes is that member functions are private by default in structures.	False
A constructor that takes no arguments is called the working constructor.	False

A class definition normally appears in a .cpp file The member function int& hours(); provides read-only access to the hours property (however it is stored) You may add = default; to the prototype of any constructor to allow the compiler to synthesize one for you The semicolon following a class definition is optional Member functions that initialize the data members of a new object are called accessors Using classes for user-defined types means that you cannot change the data representation without affecting the users of your class	
The keywords public and private are the C++ mechanism for defining interfaces and enforcing encapsulation Member functions that change the state of an object are called mutators A constructor that takes no arguments is called the default constructor Using classes for user-defined types means that you can enforce restrictions on data member access Accessor member functions are allowed to read data members and also change them	True
[2101] Which of these is not a property of an object (in the OOP sense)? Substitutablility Identity State All of these are properties of an object Behavior	Substitutablility
[2102] The _____ of an object consist of its attributes or characteristics, represented by the values stored in its data members. Identity State Class Behavior Object	State
[2103] A(n) _____ is a template or blueprint specifying the data attributes and behaviors for a group of similar objects. Behavior State Class Object Identity	Class
[2104] The _____ of an object is implemented by the object's member functions. Class Identity State Object Behavior	Behavior
[2105] Objects are _____ of a particular class. Instances Abstractions Identifiers Interfaces Encapsulations	Instances
[2106] _____ is the Object-Oriented design principle and technique that enforces data hiding. Abstraction Inheritance Dynamic Binding Polymorphism Encapsulation	Encapsulation

attributes and behaviors. Abstraction Inheritance Dynamic Binding Polymorphism Encapsulation	
[2108] _____ is the Object-Oriented design feature that allows you to write programs in terms of an "ideal" class, but substitute or plug-in related objects that act in different ways when your program runs. Inheritance Polymorphism Dynamic Binding Abstraction Encapsulation	Polymorphism

[2109] The working constructor for this class is:
 	
class Radio {
public:
Radio();
explicit Radio(double);
Radio(double, int);
 	
double frequency() const;
double frequency(double);
};
Radio(double, int)
Radio(double)
Radio()
None of these
	Radio(double, int)

[2110] The conversion constructor for this class is:
 	
class Radio {
public:
Radio();
explicit Radio(double);
Radio(double, int);
 	
double frequency() const;
double frequency(double);
};
None of these
Radio(double)
Radio(double, int)
Radio()
	Radio(double)

[2111] The copy constructor for this class is:
 	
class Radio {
public:
Radio();
explicit Radio(double);
Radio(double, int);
 	
double frequency() const;
double frequency(double);
};
Radio(double, int)
None of these
Radio(double)
Radio()
	None of these

[2112] What statement about constructors is false? All constructors are passed a pointer argument Constructors have no return type You must write at least one constructor for every class Constructors may take arguments Classes may have more than one constructor	You must write at least one constructor for every class
[2113] Which members often use the modifier explicit in their declaration? The copy constructor The conversion constructor The default constructor None of these The working constructor	The conversion constructor

#include <string>
class Xynoid {
double a;
int b;
std::string c;
};

int main()
{
Xynoid x;
}

Does not compile.
Compiles and links. Two members uninitialized
Compiles and links. All members initialized.
Compiles but does not link.
Compiles and links. All members uninitialized

[2115] The following code:

#include <string>
class Xynoid {
double a{3.14};
int b = 42;
std::string c;
};

int main()
{
Xynoid x;
}

Compiles and links. All members uninitialized
Does not compile.
Compiles and links. Two members uninitialized
Compiles and links. All members initialized
Compiles but does not link.

Compiles and links. All members initialized

[2116] The following code:

#include <string>
class Xynoid {
double a{3.14};
int b = 42;
std::string c;
public:
Xynoid(double x, int y, std::string z);
};

int main()
{
Xynoid x;
}

Compiles and links. Two members uninitialized
Compiles but does not link.
Compiles and links. All members uninitialized
Compiles and links. All members uninitialized
Does not compile.

Does not compile.

[2117] The following code:

#include <string>
class Xynoid {
double a{3.14};
int b = 42;
std::string c;
public:
Xynoid() = default;
Xynoid(double x, int y, std::string z);
};

int main()
{
Xynoid x;
Xynoid z(1, 2, "fred");
}

Does not compile.
Compiles and links. All members uninitialized
Compiles but does not link.
Compiles and links. All members initialized
Compiles and links. Two members uninitialized

Compiles but does not link

#include <string>
class Xynoid {
double a{3.14};
int b = 42;
std::string c;
public:
Xynoid() = default;
Xynoid(double x, int y, std::string z);
};

Xynoid::Xynoid(double x, int y, std::string z)
: c(z), b(y), a(x) { }

Constructor parameters are in the wrong order
Initializers use the wrong parameter values
There is no error. It is fine.
Initializers are in the wrong order.
There is no code in the body of the constructor

[2119] What happens here?

#include <iostream>
using namespace std;

class Dog {
int age_= 7;
public:
Dog(int a);
int get() const;
};
Dog::Dog(int a): age_(a) { }
int Dog::get() const { return age_; }

int main()
{
Dog a(5);
Dog b(a);
Dog c = 10;
cout << a.get() << b.get() << c.get() << endl;
}

Line Dog b(a); does not compile. No suitable constructor.
Segmentation fault when line Dog c = 7 run.
Compiles, links: prints 5510
Compiles, links: prints 5710
Line Dog c = 10; do

Compiles, links: prints 5510

[2120] What happens here?

#include <iostream>
using namespace std;

class Dog {
int age_= 7;
public:
explicit Dog(int a);
int get() const;
};
Dog::Dog(int a): age_(a) { }
int Dog::get() const { return age_; }

int main()
{
Dog a(5);
Dog b(a);
Dog c = 10;
cout << a.get() << b.get() << c.get() << endl;
}

Line Dog b(a); does not compile. No suitable constructor.
Segmentation fault when line Dog c = 7 run.
Line Dog c = 10; does not compile. Wrong type used for initializer.
Compiles, links: prints 5710
Compiles, links: prints 5510

Line Dog c = 10; does not compile. Wrong type used for initializer.

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

class Cat {
string name_;
public:
Cat(const string& n);
string get() const;
};

Cat::Cat(const string& n): name_(n) {}
string Cat::get() const { return name_; }

int main()
{
string s = "Bill";
Cat b;
b = s;
cout << b.get() << endl;
}
```

Line beginning with: string Cat::get should not have const in implementation.
Line Cat b; does not compile. No suitable constructor.
Line beginning with: Cat::Cat should not have empty body
Line b = s; does not compile. Type mismatch
The does compile; it prints "Bill".

[2122] What happens with this code?

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

class Cat {
string name_;
public:
Cat();
Cat(const string& n);
string get() const;
};

Cat::Cat(const string& n): name_(n) {}
string Cat::get() const { return name_; }

int main()
{
string s = "Bill";
Cat b;
b = s;
cout << b.get() << endl;
}
```

The does compile; it prints "Bill".
Line beginning with: Cat::Cat should not have empty body
Line b = s; does not compile. Type mismatch
Line Cat b; does not compile. No suitable constructor.
Line Cat b; does not link. No suitable implementation.

Line Cat b; does not link. No suitable implementation.

[2124] What happens with this code?

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

class Cat {
string name_;
public:
Cat() = default;
explicit Cat(const string& n);
string get() const;
};

Cat::Cat(const string& n): name_(n) {}
string Cat::get() const { return name_; }

int main()
{
string s = "Bill";
Cat b;
b = s;
cout << b.get() << endl;
}
```

Line beginning with: Cat::Cat should not have empty body
The does compile; it prints "Bill".
Line Cat b; does not compile. No suitable constructor.
Line Cat b; does not link. No suitable implementation.
Line b = s; does not compile. Type mismatch

Line b = s; does not compile. Type mismatch

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

class Cat {
string name_;
public:
Cat() = default;
explicit Cat(const string& n);
string get() const;
};

explicit Cat::Cat(const string& n): name_(n) {}
string Cat::get() const { return name_; }

int main()
{
string s = "Bill";
Cat b;
b = s;
cout << b.get() << endl;
}
```

Line beginning with: explicit Cat::Cat should not repeat explicit in implementation
Line beginning with: string Cat::get should not repeat const in implementation
Line Cat b; does not compile. No suitable constructor.
Line b = s; does not compile. Type mismatch
The does compile; it prints "Bill".

[2126] What happens with this code?

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

class Cat {
string name_;
public:
Cat() = default;
explicit Cat(const string& n);
string get() const;
};

Cat::Cat(const string& n): name_(n) {}
string Cat::get() { return name_; }

int main()
{
string s = "Bill";
Cat b;
b = s;
cout << b.get() << endl;
}
```

Line beginning with: string Cat::get should repeat const in implementation
Line b = s; does not compile. Type mismatch
The does compile; it prints "Bill".
Line Cat b; does not compile. No suitable constructor.
Line beginning with: Cat::Cat should not have an empty body.

Line beginning with: string Cat::get should repeat const in implementation

A behavior of an object is represented by the messages that it responds to.	True
Using encapsulation such as that used with structures, risks accidental data corruption.	False
Constructors always have the same name as the class, except that the constructor name is capitalized.	False
In C++11 you can initialize members in the initializer list using braces, parentheses or the assignment operator syntax.	False
Inheritance enforces the principle of data hiding.	False
Constructors must be explicitly called after an object is created.	False
Constructors are called implicitly whenever an object is created.	True
In a constructor, initializing data members by using the assignment operator means that those objects may be initialized twice.	True
The constructor that is used to initialize all of an object's fields is called the working constructor.	True
Constructors always have the same name as the class and a return type of void.	False
Initialization of data members occurs according to the order they are listed in the class definition.	True

The state of an object refers to the names and types of its data members.	False
With inheritance, the class you build upon is called a base class in C++.	True
With inheritance, the class you build upon is called a superclass in C++.	False
Suppose you have two classes related by inheritance: Dog and Poodle. According to the principle of substitutability, a function void walk(Poodle& p) will accept a Dog as an argument.	False
The state of an object refers to the combination of values stored in its data members.	True
Creating objects from a class is called instantiation.	True
If you do not create a constructor for your class, C++ will synthesize a working constructor for you.	False
The attributes of an object refers to the names and types of its data members.	True
If you do not create a constructor for your class, C++ will synthesize a default constructor for you.	True
A constructor is a member function whose job is to initialize an object into a well-formed state.	True
A constructor that takes a single argument of a different type is also known as a conversion constructor.	True
Polymorphism enforces the principle of data hiding.	False
A constructor that takes a single argument of a different type may be called implicitly.	True
Constructors always have the same name as the class and no return type.	True
A class specifies the behavior of the objects it creates through the definition of embedded functions, called member functions.	True
The constructor that is used to initialize all of an object's fields is called the default constructor.	False
With inheritance, the new class you create is called a base class in C++.	False
A function that is marked with the keyword inline should be placed in the implementation .cpp file.	False
Object behavior is implemented by data member.	False
Constructors always have the same name as the class, preceded by the tilde character (~).	False
A class represents a template or blueprint for creating objects of a particular kind.	True
Polymorphism only works in the presence of inheritance.	True
Your class may have more than one constructor.	True
Encapsulation enforces the principle of data hiding.	True
An object (in the OOP sense) is an instance of a particular class.	True
Object behavior is implemented by member functions.	True
A function that is marked with the keyword inline must be places in the header file.	True
C++11 you can ask the compiler to retain the synthesized constructor when adding new ones.	True
With inheritance, the class you build upon is called a derived class in C++.	False
The constructor that takes no arguments is called the working constructor.	False
Although not possible in earlier versions of C++, in C++11 you can ask the compiler to retain the synthesized constructor when adding new ones.	True
With inheritance, the new class you create is called a derived class in C++.	True
Objects are variables of programmer-defined types.	True
Marking a constructor with the explicit keyword, prevents unintended conversions.	True

With inheritance, the new class you create is called a subclass in C++.	False
In a constructor, objects can be initialized immediately before the opening brace of the constructor, before any other code has been run.	True
In a constructor, objects can be initialized immediately after the opening brace of the constructor, before any other code has been run.	False
With inheritance, a family of related classes is called a class hierarchy.	True
A reference variable has the same identity as the variable it refers to.	True
The constructor that takes no arguments is called the default constructor.	True
A reference variable has a different identity than the variable it refers to.	False
Suppose you have two classes related by inheritance: Dog and Poodle. According to the rules of inheritance, Poodle is a specialization of Dog.	True
The constructor initializer list is preceded by a colon and followed by a semicolon.	False
A class specifies the attributes of the objects it creates through the definition of internal data members.	True
Polymorphism means that different objects (of different types) can respond to the same message in different ways.	True
Suppose you have two classes related by inheritance: Dog and Poodle. According to the rules of inheritance, Dog is a specialization of Poodle.	False
With inheritance, the class you build upon is called a subclass in C++.	False
The attributes of an object refers to the combination of values stored in its data members.	False
The constructor initializer list follows the parameter list and precedes the constructor body.	True
Suppose you have two classes related by inheritance: Dog and Poodle. According to the principle of substitutability, a function void walk(Dog& d) will accept a Poodle as an argument.	False
When not using encapsulation, such with structures, you risk accidental data corruption.	True
Initialization of data members occurs according to the order they are listed in the initializer list.	False
To ask a particular object to perform a particular action, you send your request by calling a member function.	True
When not using encapsulation, such with structures, changing the implementation of the structure changes the interface as well.	True
With inheritance, the new class you create is called a superclass in C++.	False
A class specifies the attributes of the objects it creates through the definition of embedded functions, called member functions.	False
[2201] The BigNum class allows you to create arbitrarily large numbers, without any approximations. Assume you have the following code. What is the best header for the required operator? BigNum a{"12345.795"}, b{".95873421"}; auto c = a * b; const BigNum operator*(const BigNum& lhs, const BigNum& rhs); BigNum BigNum::operator*(const BigNum& rhs) const; BigNum& operator*(const BigNum& lhs, const BigNum& rhs); BigNum operator*(const BigNum& lhs, const BigNum& rhs);	const BigNum operator*(const BigNum& lhs, const BigNum& rhs);

assuming that the BigNum constructor is non-explicit? BigNum a{9.2573e27}; auto c = 100.0 / a; := const BigNum BigNum::operator/(const BigNum& rhs) const; const BigNum operator/(const BigNum& lhs, const BigNum& rhs); const BigNum operator/(double lhs, const BigNum& rhs); All of these can be used =:	
[2203] The BigNum class allows you to create arbitrarily large numbers, without loss of precision. Which of the following operators (which are all valid) cannot be used, assuming that the BigNum constructor is non-explicit? BigNum a{9.2573e27}; auto c = a / 100.0; := const BigNum BigNum::operator/(const BigNum& rhs)const; const BigNum operator/(const BigNum& lhs, const BigNum& rhs); const BigNum operator/(const BigNum& lhs, double rhs); All of these can be used =:	All of these can be used

[2204] The BigNum class allows you to create arbitrarily large numbers, without loss of precision. Examine the code shown. Which expression invokes the operator defined here?
 	
BigNum a{"12345.795"}, b{".95873421"};
const BigNum BigNum::operator-(const BigNum& n) const {...}
 	
:=
a -= b;
 	
auto c = a - b;
 	
auto c = --b;
 	
None of these
 	
auto c = -b;
=:
	auto c = a - b;

[2205] The BigNum class allows you to create arbitrarily large numbers, without loss of precision. Examine the code shown. Which expression invokes the operator defined here?
 	
BigNum a{"12345.795"}, b{".95873421"};
const BigNum operator-(const BigNum& n) {...}
 	
:=
None of these
 	
auto c = a - b;
 	
a -= b;
 	
auto c = --b;
 	
auto c = -b;
=:
	auto c = -b;

[2206] The BigNum class allows you to create arbitrarily large numbers, without loss of precision. Examine the code shown. Which expression invokes the operator defined here?
 	
BigNum a{"12345.795"}, b{".95873421"};
const BigNum operator-() {...}
auto c = a - b;
a -= b;
auto c = -b;
auto c = --b;
None of these
	None of these

defined here?
 	
BigNum a{"12345.795"}, b{".95873421"};
const BigNum operator-(const BigNum&, const BigNum&) {...}
 	
:=
None of these
 	
a -= b;
 	
auto c = a - b;
 	
auto c = --b;
 	
auto c = -b;
=:	

[2208] The Date class represents a day on a calendar. Examine the code shown. Which operator is called?
 	
Date d{2018, 7, 4};
auto e = d++;
const Date Date::operator++(int);
const Date Date::operator++();
Date& Date::operator++();
 	
Date& Date::operator++(int);
	const Date Date::operator++(int);

[2209] The Date class represents a day on a calendar. Examine the code shown. Which operator is called?
 	
Date d{2018, 7, 4};
auto e = d++;
 	
Date& operator++(Date&, int);
None of these
const Date operator++(Date&, int);
const Date operator++(Date&);
Date& operator++(Date&);
	const Date operator++(Date&, int);

[2210] The Date class represents a day on a calendar. Examine the code shown. Which operator is called?
 	
Date d{2018, 7, 4};
auto e = ++d;
const Date Date::operator++(int);
Date& Date::operator++(int);
None of these
const Date Date::operator++();
Date& Date::operator++();
	Date& Date::operator++();

[2211] The Date class represents a day on a calendar. Examine the code shown. Which operator is called?
 	
Date d{2018, 7, 4};
auto e = ++d;
 	
:=
const Date operator++(Date&);
 	
Date& operator++(Date&, int);
 	
Date& operator++(Date&, int);
 	
Date& operator++(Date&);
 	
const Date operator++(Date&, int);
 	
None of these
=:
	Date& operator++(Date&);

[2214] The Time class represents the time of day on a clock. Examine the code shown. Which operator is called? Time t(8, 30, "a"); cout << t << endl; := ostream& ostream::operator<<(const Time&); ostream operator<<(ostream, Time); ostream& operator<<(ostream&, const Time&); None of these ostream&Time::operator<<(ostream&, const Time&); =:	ostream& operator<<(ostream&, const Time&);

class Point {
int x_{0}, y_{0};
public:
int x() const;
int y() const;
};

bool operator==(const Point& lhs, const Point& rhs)
{
return _____;
}
this->x() == rhs.() && this->y() == rhs.y()
None of these
lhs.x() == rhs.() && lhs.y() == rhs.y()
lhs.x_ == rhs.x_ && lhs.y_ == rhs.y_
this->x_ == rhs.x && this->y_ == rhs.y_

[2216] The Point class represents x,y coordinates in a Cartesian plane. Which line of code appears in the blank? (Members written inline for this problem.)

class Point {
int x_{0}, y_{0};
public:
int x() const { return x_; }
int y() const { return y_; }
bool operator==(const Point& rhs) const {
return _____;
}
};
x_ == rhs.x_ && y_ == rhs.y_
x() == rhs.() && y() == rhs.y()
this->x() == rhs.() && this->y() == rhs.y()
this->x_ == rhs.x && this->y_ == rhs.y_
All of these will work

All of these will work

[2217] The Point class represents x,y coordinates in a Cartesian plane. Which line of code appears completes this operator which transforms a Point by dx and dy? (Members written inline for this problem.)

class Point {
int x_{0}, y_{0};
public:
Point(int x, int y): x_{x}, y_{y} {}
int x() const { return x_; }
int y() const { return y_; }
Point operator+(int dx, int dy) const {
return _____;
}
};
Point(x() + dx, y() + dy);
Point(x_ + y_, dx + dy);
Does not compile, changes arity of operator.
Does not compile; must be a non-member function.
Does not compile; must have one user-defined type as argument.

Does not compile, changes arity of operator.

[2218] The Point class represents x,y coordinates in a Cartesian plane. Which line of code appears completes this operator which transforms a Point by dx and dy? (Members written inline for this problem.)

class Point {
int x_{0}, y_{0};
public:
Point(int x, int y): x_{x}, y_{y} {}
int x() const { return x_; }
int y() const { return y_; }
};

Point operator+(int dx, int dy) {
return _____;
}

:=
Does not compile; must have one user-defined type as argument.

Does not compile, changes arity of operator

Does not compile; must be a member function

Point(x_ + y_, dx + dy)

Point(x() + dx, y() + dy)
=:

Does not compile; must have one user-defined type as argument.

in memory? (Members written inline for this problem.)

```
class Point {
int x_{0}, y_{0};
public:
Point(int x, int y): x_{x}, y_{y} {}
int x() const { return x_; }
int y() const { return y_; }
};

Point operator@(const Point& p) {
return _____;
}
```

Does not compile; uses a non-operator symbol.
Does not compile; changes arity of operator.
Does not compile; must be a member function.
*&p
&p

[2220] The Point class represents x,y coordinates in a Cartesian plane. Which line of code appears completes this operator? (Members written inline for this problem.)

```
class Point {
int x_{0}, y_{0};
public:
Point(int x, int y): x_{x}, y_{y} {}
int x() const { return x_; }
int y() const { return y_; }
};

const Point operator++(Point& p, int n) {
Point temp(p);
. . .
return _____;
}
```

*this
Does not compile; cannot change data members of object; no mutators.
temp
Does not compile; must be a member function.
Does not compile; changes arity of operator; should be unary, not binary.

Does not compile; cannot change data members of object; no mutators.

[2221] The Point class represents x,y coordinates in a Cartesian plane. Which line of code appears completes this operator? (Members written inline for this problem.)

```
class Point {
int x_{0}, y_{0};
public:
Point(int x, int y): x_{x}, y_{y} {}
int x() const { return x_; }
int y() const { return y_; }
const Point operator++(int n) {
Point temp(*this);
. . .
return _____;
}
};
```

temp
*this
Does not compile; changes arity of operator; should be unary, not binary.
Does not compile; must be a non-member function.
Does not compile; cannot change data members of object; no mutators.

temp

[2222] The Point class represents x,y coordinates in a Cartesian plane. Which line of code appears completes this operator? (Members written inline for this problem.)

```
class Point {
int x_{0}, y_{0};
public:
Point(int x, int y): x_{x}, y_{y} {}
int x() const { return x_; }
int y() const { return y_; }
Point& operator++() {
Point temp(*this);
. . .
return _____;
}
};
```

*this

temp

Does not compile; cannot change data members of object; no mutators

Does not compile; must be a non-member function

Does not compile; changes arity of operator; should be unary, not empty

*this

class Point {
int x_{0}, y_{0};
public:
Point(int x, int y): x_{x}, y_{y} {}
int x() const { return x_; }
int y() const { return y_; }
};

Point& operator++(Point& p) {
. . .
return _____;
}

Does not compile; cannot change data members of object; no mutators.
Does not compile; changes arity of operator; should be unary, not empty.

p

Does not compile; must be a non-member function.
*this

[2224] The Point class represents x,y coordinates in a Cartesian plane. Which line of code appears completes this operator? (Members written inline for this problem.)

class Point {
int x_{0}, y_{0};
public:
Point(int x, int y): x_{x}, y_{y} {}
int x() const { return x_; }
int y() const { return y_; }
};

Point& operator+=(Point& rhs) {
x_ += rhs.x(); y_ += rhs.y();
return _____;
}

Does not compile; missing const at the end of the operator header.
*this
Does not compile; changes arity of operator; should be unary, not binary.
Does not compile; must be a member operator.

rhs

Does not compile; must be a member operator.

[2225] The Point class represents x,y coordinates in a Cartesian plane. Which line of code appears completes this operator? (Members written inline for this problem.)

class Point {
int x_{0}, y_{0};
public:
Point(int x, int y): x_{x}, y_{y} {}
int x() const { return x_; }
int y() const { return y_; }
};

Point& operator+=(Point& lhs, const Point& rhs) {
_____
return lhs;
}

:=
lhs.x_ += rhs.x(); lhs.y_ += rhs.y();

lhs.x() += rhs.x(); lhs.y() += rhs.y();

Does not compile; rhs must not be const

Does not compile; no access to private members of lhs

Does not compile; changes arity of operator; should be unary, not binary
=:

Does not compile; no access to private members of lhs.

argument. (Members written inline for this problem.)

class Point {
int x_{0}, y_{0};
public:
Point(int x, int y): x_{x}, y_{y} {}
int x() const { return x_; }
int y() const { return y_; }
Point& operator+=(const Point& rhs) {
_____
return *this;
}
};

:=
*this = rhs;

rhs.x_ += this->x_; rhs.y_ += this->y;

this->x() += rhs.x(); this->y() += rhs.y();

Does not compile; no access to private members object

x_ += rhs.x(); y_ += rhs.y();
=:

[2227] The Point class represents x,y coordinates in a Cartesian plane. What is the mistake in this operator? (Members written inline for this problem.)

class Point {
int x_{0}, y_{0};
public:
Point(int x, int y): x_{x}, y_{y} {}
int x() const { return x_; }
int y() const { return y_; }
Point& operator+=(Point& rhs) {
. . .
return *this;
}
};
Does not compile; should be a non-member function.
The operator return type should be a const Point
The parameter should be a constant reference
The operator should end with return this, not return *this.
The operator should have const at the end of the header

The parameter should be a constant reference

[2228] The Point class represents x,y coordinates in a Cartesian plane. What is the mistake in this operator? (Members written inline for this problem.)

class Point {
int x_{0}, y_{0};
public:
Point(int x, int y): x_{x}, y_{y} {}
int x() const { return x_; }
int y() const { return y_; }
Point operator+=(const Point& rhs) {
. . .
return *this;
}
};

:=
The parameter should be a non-constant reference

The operator should have const at the end of the header

The operator return type should be a Point&

The operator should end with return this, not return *this

Does not compile; should be a non-member function
=:

The operator return type should be a Point&

class Point {
int x_{0}, y_{0};
public:
Point(int x, int y): x_{x}, y_{y} {}
int x() const { return x_; }
int y() const { return y_; }
Point& operator+=(const Point& rhs) { . . . }
};

const Point operator+(Point& lhs, const Point& rhs) {
return lhs += rhs;
}
The operator should not change any of its parameters
There is no error; it works fine.
Does not compile; should be a member function.
The rhs parameter should not be const
The operator should return lhs after adding rhs to it.

[2230] The Point class represents x,y coordinates in a Cartesian plane. What is the mistake in this operator? (Members written inline for this problem.)

class Point {
int x_{0}, y_{0};
public:
Point(int x, int y): x_{x}, y_{y} {}
int x() const { return x_; }
int y() const { return y_; }
Point& operator+=(const Point& rhs) { . . . }
};

const Point operator+(const Point& lhs, const Point& rhs) {
return Point(lhs) += rhs;
}
The operator should not change any of its parameters
The operator return type should be a Point&.
The rhs parameter should not be const
Does not compile; should be a member function.
There is no error; it works fine.

There is no error; it works fine.

[2231] The Point class represents x,y coordinates in a Cartesian plane. What is the mistake in this operator? (Members written inline for this problem.)

class Point {
int x_{0}, y_{0};
public:
Point(int x, int y): x_{x}, y_{y} {}
int x() const { return x_; }
int y() const { return y_; }
Point& operator+=(const Point& rhs) { . . . }
};

Point& operator+(const Point& lhs, const Point& rhs) {
return Point(lhs) += rhs;
}

:=
There is no error; it works fine.

Does not compile; should be a member function

The rhs parameter should not be const

The operator should not change any of its parameters

The operator return type should not be a Point&.
=:

The operator return type should not be a Point&.

```
class Point {
int x_{0}, y_{0};
public:
Point(int x, int y): x_{x}, y_{y} {}
int x() const { return x_; }
int y() const { return y_; }
};

ostream& operator<<(ostream& out, const Point& p)
{
return out << '(' << p.x() << ", " << p.y() << ')';
}
```

:=
There is no error; it works fine

The Point p parameter should not be const

Does not compile; should be a member function

You must return out after writing to it. This example returns void

The data members x_ and y_ are inaccessible in a non-member function
=:

[2233] The Point class represents x,y coordinates in a Cartesian plane. What is the mistake in this operator? (Members written inline for this problem.)

```
class Point {
int x_{0}, y_{0};
public:
Point(int x, int y): x_{x}, y_{y} {}
int x() const { return x_; }
int y() const { return y_; }
};

ostream& operator<<(ostream& out, const Point& p)
{
return out << '(' << p.x_ << ", " << p.y_ << ')';
}
```

:=
The Point p parameter should not be const

Does not compile; should be a member function

There is no error; it works fine.

You must return out after writing to it. This example returns void

The data members x_ and y_ are inaccessible in a non-member function
=:

The data members x_ and y_ are inaccessible in a non-member function

[2234] The Point class represents x,y coordinates in a Cartesian plane. What is the mistake in this operator? (Members written inline for this problem.)

```
class Point {
int x_{0}, y_{0};
public:
Point(int x, int y): x_{x}, y_{y} {}
int x() const { return x_; }
int y() const { return y_; }
};

void operator<<(ostream& out, const Point& p)
{
out << '(' << p.x() << ", " << p.y() << ')';
}
```

Does not compile; should be a member function.
The data members x_ and y_ are inaccessible in a non-member function.
The Point p parameter should not be const
You must return out after writing to it. This example returns void.
There is no error; it works fine.

You must return out after writing to it. This example returns void.

class Point {
int x_{0}, y_{0};
public:
Point(int x, int y): x_{x}, y_{y} {}
int x() const { return x_; }
int y() const { return y_; }
};
 	
ostream& operator<<(ostream& out, Point& p)
{
return out << '(' << p.x() << ", " << p.y() << ')';
}
 	
The data members x_ and y_ are inaccessible in a non-member function.
There is no error; it works fine.
The Point p parameter should be const
Does not compile; should be a member function.
You must first write to out and then return it.	

The prototype for a member subtraction operator for the type T is: const T operator-(const T& lhs, const T& rhs);	False
The prototype for a non-member addition operator for the type T is: const T operator+(const T& rhs) const;	False
The prototype for a non-member subtraction operator for the type T is: const T operator-(const T& lhs, const T& rhs);	True
You may not overload the scope operator ::.	True
The parameter names lhs and rhs are commonly used with overloaded operators.	True
Overloaded operators may be implemented as member functions.	True
The expression *this can be returned from non-member operators.	False
The short-hand assignment operators for type T should return *this.	True
The expression *this can only be returned from member operators.	True
With operator overloading, you may use any symbol to define a new operator.	False
You must use the ordinary meaning of an operator when you overload it. It would be impossible to redefine subtraction to mean addition, for instance.	False
The prototype for a member addition operator for the type T is: const T operator+(const T& rhs) const;	True
To compare objects for equality, overload both == and !=.	True
Though not required, you should use the ordinary meaning of an operator when you overload it. It would be unwise to redefine subtraction to mean addition, for instance.	True
Overloaded operators are functions that use special names that begin with the keyword overloaded.	False
The arithmetic operators, such as addition, subtraction and multiplication for type T should return a const T.	True
The signature for the postfix decrement operator (of type T) is: T& operator--();	False
You may overload the conditional operator ?:.	False
The expression *this is called a self-reference.	True
Classes whose objects need to be sorted should overload == and !=.	True
You can only overload existing operators. You cannot use other symbols.	True
The signature for the postfix decrement operator (of type T) is: const T operator--(int);	True
The subscript operators must be written as a member operator.	True
You may overload operators for the built-in types.	False

The I/O operators must always be written as non-member operators.	True
Symetric operators, where the user-defined type may appear on the left or the right, should be written as member operators.	False
The short-hand assignment operators for type T should return a T&.	True
Side-effect operators, such as increment or short-hand assignment, should be written as member operators.	True
The signature for the prefix increment operator (of type T) is: const T operator++(int);	False
You may not overload the subscript ([]) operator.	False
Symetric operators, where the user-defined type may appear on the left or the right, should be written as non-member operators.	True
The arithmetic operators, such as addition, subtraction and multiplication for type T should return a T.	False
Overloaded operators may be implemented as non-member functions.	True
An overloaded operator must have at least one operand that is a user-defined type.	True
The short-hand assignment operators for type T should return a const T.	False
Member operators have direct access to private data members.	True
The I/O operators should always be written as member operators.	False
You may not overload the conditional operator ?:.	True
The parameter names left and right are commonly used with overloaded operators.	False
Classes whose objects need to be sorted should overload <.	False
The subscript operators may be written as a member operator or as a non-member operator.	False
Side-effect operators, such as increment or short-hand assignment, should be written as non-member operators.	False
Non-member operators have direct access to private data members.	True
The arithmetic operators, such as addition, subtraction and multiplication for type T should return a T&.	False
You may not overload the indirection operator, the unary *.	False

[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

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	

[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

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

15
6
10
24
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

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

[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

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

[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**ll**o"
"h**e**l**l**o*"
"h**e**l**l**o"
Stack overflow
"**h**e**ll**o"

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

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****ell**o"
"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****ell**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"

[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

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.	
[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;}

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));	

[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

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	

[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.

[2501] Below is a class hierarchy for card games. Which of the Hand member functions may be overridden in the GoFishHand class?
 	
class Hand {
std::vector<Card> cards;
public:
void add(const Card&);
Card get(size_t index) const;
virtual int score() const;
};
 	
class PokerHand : public Hand { . . . };
class BlackjackHand : public Hand { . . . };
class GoFishHand : public Hand { . . . };
 	
get()
 	
add()
 	
score()
 	
all of them
 	
none of them
	score()

[2502] Below is a class hierarchy for card games. Which is the correct signature for a function that can print the score of any playing card hand?
 	
class Hand {
std::vector<Card> cards;
public:
void add(const Card&);
Card get(size_t index) const;
virtual int score() const;
};
 	
class PokerHand : public Hand { . . . };
class BlackjackHand : public Hand { . . . };
class GoFishHand : public Hand { . . . };
 	
void printScore(Hand h);
 	
void printScore(const Hand h);
 	
void printScore(const Hand* h);
 	
void printScore(BlackjackHand& h);
 	
void printScore(const PokerHand& h);
	void printScore(const Hand* h);

```
class Hand {
std::vector<Card> cards;
public:
void add(const Card&);
Card get(size_t index) const;
virtual int score() const;
};

class PokerHand : public Hand { . . . };
class BlackjackHand : public Hand { . . . };
class GoFishHand : public Hand { . . . };

void showScore(const Hand h 🌺) {
cout << h.score() << endl;
}
. . .
PokerHand ph; . . .
showScore(ph 🌺); // what happens here?
```

The PokerHand portion of ph is sliced off and it becomes a Hand object

It does not compile because ph is not a Hand object so we have a type error

It does not compile because you should pass &ph instead of ph

The Hand object is converted to a PokerHand object implicitly

It prints the score for the PokerHand object named ph

It prints the score for the PokerHand object named ph

[2504] Below is a class hierarchy for card games. What happens when showScore() is called?

```
class Hand {
std::vector<Card> cards;
public:
void add(const Card&);
Card get(size_t index) const;
virtual int score() const;
};

class PokerHand : public Hand { . . . };
class BlackjackHand : public Hand { . . . };
class GoFishHand : public Hand { . . . };

void showScore(const Hand* h 🌺) {
cout << h.score() << endl;
}
. . .
PokerHand ph; . . .
showScore(&ph 🌺); // what happens here?
```

It calls the PokerHand::score() function if one has been defined

It does not compile because ph is not a Hand object so a pointer mismatch error

The PokerHand portion of ph is sliced off and it becomes a Hand object

It does not compile because you should pass ph instead of &ph

It calls the Hand::score() function because score() is virtual

It calls the PokerHand::score() function if one has been defined

class Hand {
std::vector<Card> cards;
public:
void add(const Card&);
Card get(size_t index) const;
virtual int score() const;
};

class PokerHand : public Hand { . . . };
class BlackjackHand : public Hand { . . . };
class GoFishHand : public Hand { . . . };

void showScore(const Hand& h 🌺) {
cout << h.score() << endl;
}
. . .
PokerHand ph; . . .
showScore(ph 🌺); // what happens here?

The PokerHand portion of ph is sliced off and it becomes a Hand object

It calls the Hand::score() function because score() is virtual

It calls the PokerHand::score() function if one has been defined

It does not compile because ph is not a Hand object so a pointer mismatch error

It does not compile
because you should pass ph instead of &ph.

[2506] Below is a class hierarchy for card games. What happens when showScore() is called?

class Hand {
std::vector<Card> cards;
public:
void add(const Card&);
Card get(size_t index) const;
int score() const;
};

class PokerHand : public Hand { . . . };
class BlackjackHand : public Hand { . . . };
class GoFishHand : public Hand { . . . };

void showScore(const Hand& h 🌺) {
cout << h.score() << endl;
}
. . .
PokerHand ph; . . .
showScore(ph 🌺); // what happens here?

It does not compile because you should pass ph instead of &ph

It calls the Hand::score() function because score() is not virtual

It calls the PokerHand::score() function if one has been defined

It does not compile because ph is not a Hand object so a pointer mismatch error

The PokerHand portion of ph is sliced off and it becomes a Hand object

It calls the Hand::score() function because score() is not virtual

---

[2507] Below is a class hierarchy for card games. Assuming that these are the only classes and that the concrete classes are correctly completed, which of the following non-member functions are polymorphic?

class Hand {
std::vector<Card> cards;
public:
void add(const Card&);
Card get(size_t index) const;
virtual int score() const;
};

class PokerHand : public Hand { . . . };
class BlackjackHand : public Hand { . . . };
class GoFishHand : public Hand { . . . };

void draw(const Hand h) { . . . }

void draw(const Hand& h) { . . . }

void draw(const PokerHand* h) { . . . }

void draw(const GoFishHand& h) { . . . }

void draw(const Hand& h) { . . . }

following non-member functions are polymorphic?

```
class Hand {
std::vector<Card> cards;
public:
void add(const Card&);
Card get(size_t index) const;
virtual int score() const;
};

class PokerHand : public Hand { . . . };
class BlackjackHand : public Hand { . . . };
class GoFishHand : public Hand { . . . };


void draw(const Hand h) { . . . }

void draw(const Hand* h) { . . . }

void draw(const PokerHand& h) { . . . }

void draw(const GoFishHand* h) { . . . }
```

[2509] Below is a class hierarchy for card games. Assuming that these are the only classes and that the concrete classes are correctly completed, which of the following definitions will not compile?

```
class Hand {
std::vector<Card> cards;
public:
void add(const Card&);
Card get(size_t index) const;
virtual int score() const;
};

class PokerHand : public Hand { . . . };
class BlackjackHand : public Hand { . . . };
class GoFishHand : public Hand { . . . };


Hand* h = new Hand;

BlackjackHand* h = new Hand;

Hand* h = new BlackjackHand;

GoFishHand gfh; Hand& h = gfh;
```

BlackjackHand* h = new Hand;

[2510] Below is a class hierarchy for card games. Assuming that these are the only classes and that the concrete classes are correctly completed, which of the following definitions will not compile?

```
class Hand {
std::vector<Card> cards;
public:
void add(const Card&);
Card get(size_t index) const;
virtual int score() const;
};

class PokerHand : public Hand { . . . };
class BlackjackHand : public Hand { . . . };
class GoFishHand : public Hand { . . . };


GoFishHand gfh;

Hand* h = new Hand;

PokerHand* = new Hand;

Hand& h = *(new PokerHand);
```

PokerHand* = new Hand;

class Pet { . . . };
class Puppy : public Pet { . . . };
class Kitty : public Pet { . . . };
class Ducky : public Pet { . . . };

Pet pet;
Puppy pup;
Kitty kit;
Duck duck;

pet = kit;

pet = pup;

Puppy& pr = pup;

Pet* p = &duck;

All of these will compile

[2512] Below is a class hierarchy. Which assignment will fail to compile?

class Pet { . . . };
class Puppy : public Pet { . . . };
class Kitty : public Pet { . . . };
class Ducky : public Pet { . . . };

Pet pet;
Puppy pup;
Kitty kit;
Duck duck;

pet = pup;

Puppy* p = &pet;

Puppy& pr = pup;

Pet* p = &duck;

Puppy* p = &pet;

[2513] Below is a class hierarchy. Which assignment will fail to compile?

class Pet { . . . };
class Puppy : public Pet { . . . };
class Kitty : public Pet { . . . };
class Ducky : public Pet { . . . };

Pet pet;
Puppy pup;
Kitty kit;
Duck duck;

pup = pet;

Pet* p = &pet;

Pet& p = duck;

Puppy& pr = pup;

pup = pet;

[2514] Below is a class hierarchy. Which assignment results in slicing?

class Pet { . . . };
class Puppy : public Pet { . . . };
class Kitty : public Pet { . . . };
class Ducky : public Pet { . . . };

Pet pet;
Puppy pup;
Kitty kit;
Duck duck;

pet = pup;

pup = pet;

Pet* p = &pet;

Pet& p = duck;

pet = pup;

class Pet { . . . };
class Puppy : public Pet { . . . };
class Kitty : public Pet { . . . };
class Ducky : public Pet { . . . };

Pet pet;
Puppy pup;
Kitty kit;
Duck duck;

pet = kit;

kit = duck;

pup = pet;

duck = pet;

[2516] Below is a class hierarchy. Which assignments are illegal?

class Widget { . . . };
class Label: public Widget { . . . };
class Button: public Widget { . . . };
class Text: public Widget { . . . };
class TextArea: public Text { . . . };
class TextLine: public Text { . . . };
class Container: public Widget { . . . };
class Canvas: public Container { . . . };
class Window: public Container { . . . };

Button* p = new Button;

Widget* p = new Window;

Widget* p = new TextLine;

Container* p = new Canvas;

None of these are illegal

None of these are illegal

[2517] Below is a class hierarchy. Which assignments are illegal?

class Widget { . . . };
class Label: public Widget { . . . };
class Button: public Widget { . . . };
class Text: public Widget { . . . };
class TextArea: public Text { . . . };
class TextLine: public Text { . . . };
class Container: public Widget { . . . };
class Canvas: public Container { . . . };
class Window: public Container { . . . };

Text* p = new TextArea;

Widget* p = new Window;

Canvas* p = new Container;

None of these are illegal

Widget* p = new Widget;

Canvas* p = new Container;

[2518] Below is a class hierarchy. Which assignments are illegal?

class Widget { . . . };
class Label: public Widget { . . . };
class Button: public Widget { . . . };
class Text: public Widget { . . . };
class TextArea: public Text { . . . };
class TextLine: public Text { . . . };
class Container: public Widget { . . . };
class Canvas: public Container { . . . };
class Window: public Container { . . . };

None of these are illegal

Widget* p = new Canvas;

Window* p = new Container;

Text* p = new TextLine;

Widget* p = new TextArea;

Window* p = new Container;

```
class Widget { . . . };
class Label: public Widget { . . . };
class Button: public Widget { . . . };
class Text: public Widget { . . . };
class TextArea: public Text { . . . };
class TextLine: public Text { . . . };
class Container: public Widget { . . . };
class Canvas: public Container { . . . };
class Window: public Container { . . . };
```

```
Text p = TextLine();
```

```
Widget p = Widget();
```

```
Widget* p = new TextArea;
```

```
Container& p = *(new Window);
```

[2520] Below is a class hierarchy. Which statements may result in slicing?

```
class Writer { . . . };
class Pen : public Writer { . . . };
class Pencil : public Writer { . . . };
class FountainPen : public Pen { . . . };
```

```
Writer p = Writer ();
```

```
Pen* p = new Writer();
```

```
Pen p = FountainPen();
```

```
Writer& p = *(new Pencil);
```

```
Pen p = FountainPen();
```

[2521] What prints when this code is run?

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

class Shape {
public:
 virtual string toString() const { return "Shape"; }
};

class Circle : public Shape {
public:
string toString() const { return "Circle"; }
};

class Triangle : public Shape {
public:
string toString() const { return "Triangle"; }
};

int main() {
Shape s1;
Shape s2 = Triangle();
cout << s1.toString() << s2.toString() << endl;
}
```

Triangle

ShapeShape

ShapeTriangle

Compiles but prints something else

ShapeShape

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

class Shape {
public:
🌺 virtual string toString() const { return "Shape"; }
};

class Circle : public Shape {
public:
string toString() const { return "Circle"; }
};

class Triangle : public Shape {
public:
string toString() const { return "Triangle"; }
};

int main() {
Shape* s1 = new Circle;
Shape* s2 = new Triangle;
cout << s1->toString() << s2->toString() << endl;
}
```

ShapeShape

CircleTriangle

ShapeTriangle

Compiles but prints something else

[2523] What prints when this code is run?

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

class Shape {
public:
🌺 string toString() const { return "Shape"; }
};

class Circle : public Shape {
public:
string toString() const { return "Circle"; }
};

class Triangle : public Shape {
public:
string toString() const { return "Triangle"; }
};

int main() {
Shape* s1 = new Circle;
Shape* s2 = new Triangle;
cout << s1->toString() << s2->toString() << endl;
}
```

ShapeShape

ShapeShape

ShapeTriangle

CircleTriangle

Does not compile

ShapeShape

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

class Shape {
public:
🌺 string toString() const { return "Shape"; }
};

class Circle : public Shape {
public:
virtual string toString() const { return "Circle"; }
};

class Triangle : public Shape {
public:
virtual string toString() const { return "Triangle"; }
};

int main() {
Shape* s1 = new Circle;
Shape* s2 = new Triangle;
cout << s1->toString() << s2->toString() << endl;
}

ShapeShape

ShapeTriangle

CircleTriangle

Does not compile

[2525] What prints when this code is run?

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

class Shape {
public:
virtual string toString() const { return "Shape"; }
};

class Circle : public Shape {
public:
virtual string toString() const { return "Circle"; }
};

class Triangle : public Shape {
public:
virtual string toString() const { return "Triangle"; }
};

int main() {
Shape s1 = Circle();
Shape* s2 = new Triangle;
cout << s1.toString() << s2->toString() << endl;
}

ShapeShape

ShapeTriangle

CircleTriangle

Compiles but prints something else

ShapeTriangle

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

struct B { virtual string str() const { return "B"; }};
struct D1 : public B { string str() const { return "D1"; }};
struct D2 : public B { string str() const { return "D2"; }};
struct D3 : public D1 { string str() const { return "D3"; }};

int main() {
B p1(new D1), p2(new D2), *p3(new D3);
cout << p1->str() << p2->str() << p3->str() << endl;
}
```

BBB

BBD3

D1D2D3

Compiles but prints something else

[2527] What prints when this code is run? (Note that struct is used instead of class only to make all members public and to make the code shorter).

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

struct B { 🌺string str() const { return "B"; }};
struct D1 : public B { virtual string str() const { return "D1"; }};
struct D2 : public B { string str() const { return "D2"; }};
struct D3 : public D1 { string str() const { return "D3"; }};

int main() {
B p1(new D1), p2(new D2), *p3(new D3);
cout << p1->str() << p2->str() << p3->str() << endl;
}
```

BBB

BBD3

D1BD3

D1D2D3

BBB

[2528] What prints when this code is run?

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

class Shape { public: virtual void iam() const; };
class Square : public Shape { public: void iam() const; };
class Oval: public Shape { public: void iam() const; };

void Shape::iam() const { cout << "Shape"; }
void Square::iam() const { cout << "Square"; }
void Oval::iam() const { cout << "Oval"; }

void iam(Shape s) { s.iam(); } 🌺

int main() {
iam(Shape());
iam(Square());
iam(Oval());

cout << endl;
}
```

ShapeSquareShape

ShapeSquareOval

ShapeShapeOval

ShapeShapeShape

Does not compile

ShapeShapeShape

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

class Shape { public: virtual void iam() const; };
class Square : public Shape { public: void iam() const; };
class Oval: public Shape { public: void iam() const; };

void Shape::iam() const { cout << "Shape"; }
void Square::iam() const { cout << "Square"; }
void Oval::iam() const { cout << "Oval"; }

void iam(const Shape& s) { s.iam(); } 🌺

int main() {
iam(Shape());
iam(Square());
iam(Oval());
cout << endl;
}
```

ShapeShapeOval

ShapeSquareOval

ShapeShapeShape

ShapeSquareShape

[2530] What prints when this code is run?

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

class Shape { public: virtual void iam() const; };
class Square : public Shape { public: void iam() const; };
class Oval: public Shape { public: void iam() const; };

void Shape::iam() const { cout << "Shape"; }
void Square::iam() const { cout << "Square"; }
void Oval::iam() const { cout << "Oval"; }

void iam(const Shape& s) { s.iam(); } 🌺

int main() {

iam(Shape());
iam(Square());
iam(Oval());
cout << endl;
}
```

ShapeShapeOval

ShapeSquareOval

ShapeShapeShape

ShapeSquareShape

ShapeSquareOval

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

class Shape { public: virtual void iam() const; };
class Square : public Shape { public: void iam() const; };
class Oval: public Shape { public: void iam() const; };

void Shape::iam() const { cout << "Shape"; }
void Square::iam() const { cout << "Square"; }
void Oval::iam() const { cout << "Oval"; }

void iam(const Shape& s) { s.iam(); }

int main() {
Shape a = Shape(), b = Square(), c = Oval(); ..// Slices
iam(a);
iam(b);
iam(c);
}

ShapeShapeOval

ShapeSquareOval

ShapeShapeShape

ShapeSquareShape

[2532] Which member function(s) may 🌺 be overridden in Hobbit?

class Creature {
public:
Creature(const string& name);
virtual string name() const final;
virtual string skills() const;
virtual void addSkill(const string& skill);
void print() const;
};

class Hobbit : public Creature {
. . .
};

None of them

addSkill(), skills()

addSkill(), skills(), print()

addSkill(), skills(), name()

addSkill(), skills()

[2533] What prints when this code is run?

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

class Shape { public: virtual void iam() const; };
class Square : public Shape { };
class Oval: public Shape { public: void iam() const; };

void Shape::iam() const { cout << "Shape"; }
void Oval::iam() const { cout << "Oval"; }

void iam(const Shape* s) { s->iam(); }

int main() {
Shape **a = new Shape,** b = new Square, *c = new Oval;
iam(a);
iam(b);
iam(c);
}

ShapeShapeOval

ShapeSquareOval

ShapeShapeShape

ShapeSquareShape

ShapeShapeOval

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

class Shape { public: virtual void iam() const; };
class Square : public Shape { public: void iam() const; };
class Oval: public Shape { public: void iam() const; };

void Shape::iam() const { cout << "Shape"; }
void Square::iam() const { cout << "Square"; }
void Oval::iam() const { cout << "Oval"; }

void iam(const Shape& s) { s.iam(); }

int main() {
vector<Shape& 🌺> v = {Shape(), Square(), Oval()};
for (auto& e : v) iam(e);
cout << endl;
}
```

ShapeShapeOval

ShapeSquareOval

ShapeShapeShape

ShapeSquareShape

Does not compile

[2535] What prints when this code is run?

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

class Shape { public: virtual void iam() const; };
class Square : public Shape { public: void iam() const; };
class Oval: public Shape { public: void iam() const; };

void Shape::iam() const { cout << "Shape"; }
void Square::iam() const { cout << "Square"; }
void Oval::iam() const { cout << "Oval"; }

void iam(const Shape* s) { s->iam(); }

int main() {
vector<Shape* 🌺> v = {new Shape, new Square, new Oval};
for (auto& e : v) iam(e);
cout << endl;
}
```

ShapeShapeOval

ShapeSquareOval

ShapeShapeShape

ShapeSquareShape

Does not compile

ShapeSquareOval

#include <string>
#include <iostream>
#include <vector>
using namespace std;

class Shape { public: virtual void iam() const; };
class Square : public Shape { public: void iam() const; };
class Oval: public Shape { public: void iam() const; };

void Shape::iam() const { cout << "Shape"; }
void Square::iam() const { cout << "Square"; }
void Oval::iam() const { cout << "Oval"; }

void iam(const Shape& s) { s.iam(); }

int main() {
vector<Shape 🌺> v = {Shape(), Square(), Oval()};
for (auto& e : v) iam(e);
cout << endl;
}

ShapeShapeOval

ShapeSquareOval

ShapeShapeShape

ShapeSquareShape

Does not compile

[2537] What prints when this code is run?

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

class Shape { public: virtual void iam() const; };
class Square : public Shape { public: void iam() const; };
class Oval: public Shape { public: void iamm() const; }; 🌺

void Shape::iam() const { cout << "Shape"; }
void Square::iam() const { cout << "Square"; }
void Oval::iamm() const { cout << "Oval"; }

void iam(const Shape* s) { s->iam(); }

int main() {
Shape **a = new Shape,** b = new Square, *c = new Oval;
iam(a);
iam(b);
iam(c);
}

ShapeShapeOval

ShapeSquareOval

Does not compile

ShapeShapeShape

ShapeSquareShape

ShapeSquareShape

[2538] Which member function(s) must 🌺 be overridden in Hobbit?

class Creature {
public:
Creature(const string& name);
virtual string name() const final;
virtual string skills() const;
virtual void addSkill(const string& skill);
void print() const;
};

class Hobbit : public Creature {
. . .
};

None of them

addSkill(), skills()

addSkill(), skills(), print()

addSkill(), skills(), name()

None of them

class Creature {
public:
Creature(const string& name);
virtual string name() const final;
virtual string skills() const;
virtual void addSkill(const string& skill);
void print() const;
};

class Hobbit : public Creature {
. . .
};

None of them

name(), print()

skills(), name(), print()

addSkill(), skills(), name()

[2540] Which member function(s) in Hobbit cause a compiler error?

class Creature {
public:
Creature(const string& name);
virtual string name() const final;
virtual string skills() const;
virtual void addSkill(const string& skill);
void print() const;
};

class Hobbit : public Creature {
public:
string name() const override;
string skills() const override;
void addSkill(const string&) override;
void print() override;
};

None of them

name(), print()

skills(), name(), print()

addSkill(), skills(), name()

name(), print()

Tell the compiler that you intend to override a base class function by adding the keyword override as an annotation before the function header

Putting the keyword final at the end of a non-virtual member function heading prohibits derived classes from overriding that function

Virtual functions invoked through an object use late binding to decide which function to call

The composition relationship is informally known as is-a

Virtual member functions are implemented by adding a new pointer, called a vtable, to every object that contains at least one virtual function

If you make a class final then you must make all of its member functions final as well

In private inheritance derived classes inherit the interface of the base class, but not its implementation

The public inheritance relationship is informally known as implemented-with

If a derived class redefine a non-virtual base-class function it causes a syntax error

The public inheritance relationship is informally known as has-a

Non-virtual functions always use late binding to decide which function to call

Waiting until runtime to determine which function to call is known as early binding

False

The public inheritance relationship is informally known as is-a The composition relationship is informally known as has-a The keyword override allows the compiler to ensure that the base-class function you are overriding is virtual Non-virtual functions always use early, or compile-time binding to decide which function to call Creating a new class by combining instances of simpler classes as data members is called composition It is always a logic error for a derived class to redefine a non-virtual function Putting the keyword final at the end of the class heading prohibits the creation of subsequent derived classes Waiting until runtime to determine which function to call is known as late binding Waiting until runtime to determine which function to call is known as dynamic dispatch Virtual functions invoked through a pointer to a base-class object use late binding to decide which function to call If a virtual member function does not use the keyword final, then any derived class may override that function In private inheritance derived classes inherit the implementation of the base class, but not its interface Virtual functions invoked through a reference to a base-class object use late binding to decide which function to call Virtual member functions are implemented by adding a new pointer to every object that contains at least one virtual function In private inheritance a using declaration is employed to selectively bring base class members into the derived class scope Tell the compiler that you intend to override a base class function by adding the keyword override to the end of the member function declaration Putting the keyword final at the end of a virtual member function heading prohibits derived classes from overriding that function	
[2401] _____ is one of the primary mechanisms that we use to understand the natural world around us. Starting as infants we begin to recognize the difference between categories like food, toys, pets, and people. As we mature, we learn to divide these general categories or classes into subcategories like siblings and parents, vegetables and dessert classification specialization generalization encapsulation	classification
[2402] _____—the specification of attributes and behavior as a single entity—allows us to build on our understanding of the natural world as we create software encapsulation generalization inheritance polymorphism	encapsulation
[2403] Inheritance gives your programs the ability to express _____ between classes dependencies composition encapsulation relationships	relationships

specialized class, generalized class derived class, base class base class, derived class concrete class, abstract class	
[2405] A classification hierarchy represents an organization based on _____ and _____. encapsulation and polymorphism abstraction and generalization abstraction and encapsulation generalization and specialization specialization and encapsulation	generalization and specialization
[2406] When you create your own new, user-defined types, there are three different strategies you can use. Which of these is not one of those strategies? defining a class from scratch extending an existing class by adding new features combining simpler classes to create a new classes modifying an existing class	modifying an existing class
[2407] A(n) _____ relationship exists between two classes when one class contains data members that are instances of the other class Is-A Implemented-As Has-A Uses-A	Has-A
[2408] The ostream class is the/a _____ class of ios derived ancestor sibling descendent base	derived
[2409] The ostream class is the/a _____ class of ofstream descendent ancestor sibling base derived	base
[2410] The ostream class is the/a _____ class of istream ancestor derived descendent sibling base	sibling

derived base descendent ancestor sibling	
[2412] The fstream class is the/a _____ class of istream ancestor derived sibling base descendent	descendent

[2413] Which of these is an example of the principle of substitutability?
 	
void f1(fstream& out) { . . .}
void f2(int n) { . . . }
void f3(const string& s) { . . . }
void f4(ios& i) { . . . }
 	
f4(cout);
 	
f1(cout);
 	
None of these
 	
f2(3.5);
 	
f3("hello");
	f4(cout);

[2414] Which of these is an example of the principle of substitutability?
 	
void f1(ostream& out) { . . .}
void f2(double n) { . . . }
void f3(const char * s) { . . . }
void f4(ofstream& i) { . . . }
 	
f2(3);
 	
ostringstream out; f1(out);
 	
None of these
 	
f4(cout);
 	
f3("hello");
	ostringstream out; f1(out);

[2415] Assume you have a Student object named bill. Which of these statements would be legal?
 	
bill.name = "Bill Gates"; // I
bill.setName("Bill Gates"); // II
cout << bill.getName(); // III
bill.studentID = 123L; // IV
cout << bill.getID(); // V
 	
II, III, IV, V
 	
IV and V
 	
II, III, V
 	
All of them
 	
None of them
	II, III, V

name = "Bill Gates"; // I
setName("Bill Gates"); // II
name = name.substr(1); // III
studentID = 123L; // IV
studentID = getID() * 2; // V

II, IV and V

II, III, V

II, III, IV, V

All of them

None of them

None of them

[2417] Which of these data members or member functions are inherited by the Person class?

getName(), setName(), studentID, getID()

None of them

name, getName(), setName(), getID()

getName(), setName(), name

studentID, name, getName(), setName(), getID()

getName(), setName()

[2418] Which of these data members or member functions are inherited (and accessible) by the Student class?

name, getName(), setName(), getID()

getName(), setName(), name

studentID, name, getName(), setName(), getID()

getName(), setName()

None of them

name

[2419] Which of these data members or member functions are inherited but not directly accessible by the Student class?

getID()

studentID

name

setName()

getName()

Both data and behavior

[2420] What does a derived class inherit from its base class?

Only data

Neither data nor behavior

Only behavior

Both data and behavior

Model similar objects with different behavior

[2421] What is the primary purpose of inheritance?

Model one-to-many relationships between different types of objects

Model different objects which share similar performance goals

Model similar objects with different data values

Model similar objects with different behavior

derived from Question. Which of the following is true? NumericQuestion contains a numerical answer but not a query NumericQuestion contains a query and a numerical answer but no answer string It is impossible to know without examining the definition of the NumericQuestion class NumericQuestions contains both a query and an answer string.	
[2423] Which one of the following is an example of the "substitution principle"? A base-class object must be used in place of a derived class object A base-class object can be used in place of a derived class object A derived class object must be used in place of a base-class object A derived class object can be used in place of a base-class object	A derived class object can be used in place of a base-class object.
[2424] Suppose that we have a function that registers a Vehicle object. We also have a Car object that is a specialized Vehicle (defined by inheritance). The substitution principle states _____. The Car object can never be used in any function that is written to use a Vehicle object. A new registration function that is written to use a Car object can be used in place of the Vehicle registration function The Car object can be used in the Vehicle registration function because it is a kind of Vehicle The Vehicle object can always be used wherever a Car object is expected	The Car object can be used in the Vehicle registration function because it is a kind of Vehicle.
[2425] The Department of Motor Vehicles uses a vehicle registration program that declares a Vehicle class as a base class. The Car class and the Truck class both inherit from the Vehicle class. Which types of objects can be passed to the function register(Vehicle& v)? It is impossible to know without examining the implementation of the Car and Truck classes Vehicle, Car and Truck objects Only Car and Truck objects Only Vehicle objects	Vehicle, Car and Truck objects
[2426] Consider the following classes. The Vehicle class is a base class. The Car, Truck, and Motorcycle class inherit from the Vehicle class. The Sedan and SUV classes inherit from the Car class. Which of the following lists all the types of objects that cannot be passed into the function calculate_registration_fee(Car& car)? Motorcycle, Truck, and Vehicle objects Motorcycle, Truck, and Vehicle objects Motorcycle, and Truck objects Sedan and SUV objects Sedan, SUV, and Car objects	Motorcycle, Truck, and Vehicle objects
[2427] How can a derived class override a base class function? Nothing is required in the derived class - this is automatically provided by inheritance It is impossible for the derived class to override a base class function By providing a new implementation for a function, tagged with the override reserved word By providing a new implementation for a function with the same name and parameter types	By providing a new implementation for a function with the same name and parameter types

class Car
{
public:
Car();
virtual void setSpeed(double newSpeed);
double getSpeed() const;
private:
double speed;
};

The AeroCar class must define the function void override(string setSpeed, double newSpeed);

The AeroCar class must define the function void overrideSetSpeed(double)

The AeroCar class must define the function void setSpeed(double)

The AeroCar class cannot override the setSpeed member function.

[2429] What is the output?

```
class Car {
public:
virtual void setSpeed(double s) { speed = s; }
double getSpeed() const { return speed; }
private:
double speed = 0;
};

class AeroCar : public Car {
public:
void setSpeed(double s) { Car::setSpeed(10 * s); }
void addSpeed(double s) { Car::setSpeed(getSpeed() + s); }
};

int main() {
AeroCar ac1;
ac1.setSpeed(10);
ac1.addSpeed(250);
cout << "Speed: " << ac1.getSpeed();
}
```

Speed: 260

Speed: 350

Speed: 250

Speed: 420

Speed: 350

[2430] The Pet base class defines void setName(const string&). Cat is derived from Pet, but does not define setName(). What is true?

Cat class inherits the setName function

setName() cannot be called on Cat objects

Cat overrides the setName function

The Cat class will not compile because it does not define setName

Cat class inherits the setName function

[2431] Which member function from the Question class is overridden in the ChoiceQuestion class?

```
class Question {
public:
virtual void setText(const string&);
virtual void setAnswer(const string&);
virtual void display() const;
};

class ChoiceQuestion : public Question {
public:
void setText(const string&);
void setAnswer(int, const string&);
void display(const string&) const;
};
```

setText()

Question()

display()

setAnswer()

setText()

class Car {
public:
Car();
virtual void setSpeed(double);
double getSpeed() const;
void display() const;
};

class AeroCar : public Car {
public:
AeroCar();
void setSpeed(double);
void setHeight(double);
double getHeight() const;
};

Neither A nor B

car.getSpeed() and aero.getSpeed()

car.getHeight() and aero.getHeight()

Both A and B

[2433] Based on the following declaration of the Employee class where Manager is derived from Employee, which of the following are true?

class Employee {
public:
Employee();
Employee(const string&);
Employee(double);
Employee(const string&, double);
void setName(const string&);
string getName()const;
private:
string name;
double salary;
};

The Manager class does not inherit the private data members

The Manager class inherits name and salary, but Manager functions can only change the values of the name data member

A Manager object has direct access to the name and salary inherited data members

The Manager class inherits name and salary, but Manager functions cannot change the values of either data member.

The Manager class inherits name and salary, but Manager functions can only change the values of the name data member

---

[2434] The Car class inherits from the Vehicle class. The Car class contains one constructor which does not call a particular Vehicle constructor. Which of the following is true?

Vehicle constructors can never be called by the Car constructors

The Car class will not compile because it does not explicitly call a Vehicle constructor

The Vehicle default constructor is implicitly called by the Car constructor

All Vehicle constructors are implicitly called by the Car constructor.

The Vehicle default constructor is implicitly called by the Car constructor

---

[2435] The Manager class is derived from the Employee class. Manager defines a constructor, but does not explicitly call an Employee constructor. Which constructor is called by the Manager constructor?

class Employee {
public:
Employee();
Employee(const string&);
Employee(double);
Employee(const string&, double);
};

Employee(const string&, double);

Employee();

Employee(const string&);

Employee(double);

Employee();

AeroCar acar1(2000.0, 200.0);

```
class Car {
public:
Car();
Car(double);
void setSpeed(double);
double getSpeed() const;
};

class AeroCar : public Car {
public:
AeroCar();
AeroCar(double);
AeroCar(double, double);
void setHeight(double);
double getHeight() const;
};

AeroCar::AeroCar(double h, double s)
: Car(s), height(h) { }

int main() {
AeroCar acar(2000.0, 200.0);
}
```

double getSpeed() const

Car(double)

void setSpeed(double)

Car()

[2437] What prints?

```
class Employee {
public:
Employee() = default;
Employee(const string& n, double s) : name(n), salary(s) {}
void setName(const string& n) { name = n; }
void setSalary(double s) { salary = s; }
string getName() const { return name; }
double getSalary() const { return salary; }
private:
string name;
double salary = 0;
};

class Manager : public Employee {
public:
Manager() = default;
Manager(double b) { bonus = b; }
Manager(const string& n, double s, double b)
: Employee(n, s), bonus(b) {}
void setBonus(double b) { bonus = b; }
void print() const;
private:
double bonus;
};

void Manager::print() const {
cout << getName() << " $ " << getSalary()
<< "; Bonus: " << bonus << endl;
}

int main() {
Manager m1;
Manager m2(1000);
Manager m3("Peter", 30000, 1000);
m2.print();
}
```

Peter $ 30000; Bonus: 1000

$ 30000; Bonus: 1000

Peter $ 0; Bonus: 1000

$ 0; Bonus: 1000

$ 0; Bonus: 1000

Every Manager constructor will implicitly call the default Employee constructor

An Employee constructor will implicitly call the default Manager constructor

A Manager constructor can pass data to an Employee constructor

All of the above statements are true

[2439] Which among the following is the legal way of implementing the constructor of the Manager class that passes parameters to a base-class constructor?

```
class Employee {
public:
Employee();
Employee(const string&);
Employee(double);
Employee(const string&, double);
private:
string name;
double salary;
};
class Manager : public Employee {
public:
Manager();
Manager(const string& d, const string& n, double s);
private:
string department;
};


Manager::Manager(const string& d, const string& n, double s)
: Employee(s, n) { department = d; }

Manager::Manager(const string& d, const string& n, double s)
: Employee() { department = d; }

Manager::Manager(const string& d, const string& n, double s)
: Employee(n, d) { department = d; }

Manager::Manager(const string& d, const string& n, double s)
: Employee(n, s) { department = d; }
```

```
Manager::Manager(const string& d, const string& n, double s)
: Employee(n, s) { department = d; }
```

[2440] What prints here?

```
class Car {
public:
Car() = default;
Car(double s): speed(s) {}
double getSpeed() const { return speed; }
private:
double speed = 0;
};

class AeroCar : public Car {
public:
AeroCar() = default;
AeroCar(double h, double s) : Car(s * 2), height(h) {}
void display() const;
private:
double height = 0;
};
void AeroCar::display() const {
cout << "Speed: " << getSpeed()
<< "; Height: " << height << endl;
}

int main()
{
AeroCar acar1(2000, 200);
acar1.display();
}
```

Speed: 400; Height: 2000

Speed: 0; Height: 2000

Speed: 200; Height: 2000

Speed: 0; Height: 0

```
Speed: 400; Height: 2000
```

function be called? The Car setSpeed function cannot be called from the AeroCar::setSpeed function ::setSpeed(newSpeed) Car::setSpeed(newSpeed) Car::setSpeed() super::setSpeed(newSpeed)	
[2442] ChoiceQuestion is derived from the Question base class . ChoiceQuestion overrides the display() function defined in the Question base class. Which of the following will call the base class display() function from the ChoiceQuestion display() function? display() ::display() Question::display() super::display() this->display()	Question::display()

[2443] The Manager class is derived from the Employee base class. The Manager class overrides the getSalary()function. What is wrong with the following definition of getSalary() in the Manager class?
 	
double Manager::getSalary() const
{
auto baseSalary = getSalary();
return baseSalary + bonus;
}
 	
The call to getSalary should be written as this->getSalary();
 	
The call to getSalary should be written as Employee::getSalary();
 	
The Manager class cannot call the getSalary() function in the base class
 	
The initialization of baseSalary should have been auto baseSalary = Employee::salary;
	The call to getSalary should be written as Employee::getSalary();

[2444] What is printed?
 	
class Pet {
public:
Pet(const string& n) : name(n) {}
virtual void info() { cout << "My name is " << name << "."; }
private:
string name;
};
 	
class Cat : public Pet {
public:
Cat(const string& n) : Pet(n) {}
void info() {
cout << "I am a cat. "; Pet::info();
}
};
 	
int main() {
Cat cat = Cat("Felix");
cat.info();
}
 	
I am a cat. My name is Felix.
 	
My name is Felix. I am a cat.
 	
I am a cat.
 	
My name is Felix.
	I am a cat. My name is Felix.

```
class Counter {
public:
Counter(int c) : counter(c) {}
virtual void add(int n) { counter += n; }
void display() { cout << "Count->" << counter; }
private:
int counter;
};

class DoubleCounter : public Counter {
public:
DoubleCounter(int c) : Counter(c * 2) {}
void add(int n) { Counter::add(n * 2); }
};

int main() {
DoubleCounter counter(10);
counter.add(5);
counter.display();
}
```

Counter->25

Counter->15

Counter->20

Counter->30

Counter->30

[2601] What does this code mean?

```
class X : public Y {
. . .
};
```

Each X object uses- a Y object

Every Y object is-a X object

Each X object is-implemented in terms of Y

Every X object is-a Y object

Every X object has-a Y object

Every X object is-a Y object

[2602] What does this code mean?

```
class X : Y {
. . .
};
```

Each X object is-implemented in terms of Y
Each X object uses- a Y object
Every X object is-a Y object
Every X object has-a Y object
Every Y object is-a X object

Each X object is-implemented in terms of Y

[2603] What does this code mean?

```
class X {
Y y;
. . .
};
```

Every X object is-a Y object

Every Y object is-a X object

Every X object has-a Y object

Each X object is-implemented in terms of Y

Each X object uses- a Y object

Every X object has-a Y object

class X {
double x = Y().balance();
. . .
};

Every X object has-a Y object

Every Y object is-a X object

Every X object is-a Y object

Each X object uses- a Y object

Each X object is-implemented in terms of Y

[2605] Specialization inheritance means that the derived class may add new data members and member functions, and may also _____ the virtual member functions in the base class.

hide

override

overload

cast

delete

override

[2606] Which member functions in the Performer class may not be overridden?

class Performer {
public:
void dance() const;
virtual void sing() const;
virtual void act() const = 0;
};

sing()

dance()

None can be overridden

All can be overridden

act()

dance()

[2607] Which member functions in the Performer class may be overridden (but need not be)?

class Performer {
public:
void dance() const;
virtual void sing() const;
virtual void act() const = 0;
};

act()

sing()

dance()

All can be overridden

None can be overridden

sing()

[2608] Which member functions in the Performer class must be overridden?

class Performer {
public:
void dance() const;
virtual void sing() const;
virtual void act() const = 0;
};

act()

sing()

dance()

All can be overridden

None can be overridden

act()

class Performer {
public:
void dance() const;
};

class Mime : public Performer {
public:
void dance() const;
};

final

overloaded

overridden

hidden or shadowed

Illegal (does not compile)

[2610] Which member function is called?

class Performer {
public:
virtual void sing() const;
};

class Crooner : public Performer {
public:
void sing() const;
};

int main() {
Performer* p = new Crooner;
p->sing();
}

Crooner::sing()

Performer::sing()

Neither of these

Illegal (does not compile)

Crooner::sing()

[2611] Using C++ terminology, the member Card::score() is:

class Hand {
std::vector<Card> cards;
public:
Hand() = default;
virtual ~Hand() = default;
void add(const Card&);
virtual int score() const = 0;
virtual void sort();
bool operator<(const Card& rhs) const;
};

class PokerHand : public Hand { . . . }
class BlackjackHand : public Hand { . . . }

a virtual function

an abstract method

a pure virtual function

an overridden member

an overloaded member

a pure virtual function

class Hand {
std::vector<Card> cards;
public:
Hand() = default;
virtual ~Hand() = default;
void add(const Card&);
virtual int score() const = 0;
virtual void sort();
bool operator<(const Card& rhs) const;
};

class PokerHand : public Hand { . . . }
class BlackjackHand : public Hand { . . . }

Hand h

PokerHand ph;

Hand* hp = new PokerHand;

PokerHand ph; Hand& hr = ph;

BlackjackHand* bjp = new BlackjackHand;

[2613] Examine the class hierarchy below. Assume that both derived classes are concrete and completely defined. What happens when a PokerHand object is passed to the non-member draw() function, assuming that the function makes use of the virtual functions overridden in PokerHand?

class Hand {
std::vector<Card> cards;
public:
Hand() = default;
virtual ~Hand() = default;
void add(const Card&);
virtual int score() const = 0;
virtual void sort();
bool operator<(const Card& rhs) const;
};

class PokerHand : public Hand { . . . }
class BlackjackHand : public Hand { . . . }

void draw(const Hand h) { . . . }

The code compiles but fails to link

The hand is drawn appropriately

The code does not compile because the argument is of the wrong type

Code compiles, but the parameter is treated as a Hand object, not a PokerHand, so it is not drawn correctly

Code compiles, but the parameter is treated as a Hand object, not a PokerHand, so it is not drawn correctly

---

[2615] Examine the class hierarchy below. Assume that both derived classes are concrete and completely defined. Which of the following member functions cannot be overridden in the derived classes?

class Hand {
std::vector<Card> cards;
public:
Hand() = default;
virtual ~Hand() = default;
void add(const Card&);
virtual int score() const = 0;
virtual void sort();
bool operator<(const Card& rhs) const;
};

class PokerHand : public Hand { . . . }
class BlackjackHand : public Hand { . . . }

sort()

score()

~Hand()

operator<()

operator<()

be overridden in the derived classes?

```
class Hand {
std::vector<Card> cards;
public:
Hand() = default;
virtual ~Hand() = default;
void add(const Card&);
virtual int score() const = 0;
virtual void sort();
bool operator<(const Card& rhs) const;
};

class PokerHand : public Hand { . . . }
class BlackjackHand : public Hand { . . . }
```

add()

sort()

score()

~Hand()

[2616] Examine the class hierarchy below. Assume that both derived classes are concrete and completely defined. Which of the following member functions cannot be overridden in the derived classes?

```
class Hand {
std::vector<Card> cards;
public:
Hand() = default;
virtual ~Hand() = default;
void add(const Card&);
virtual int score() const = 0;
virtual void sort();
bool operator<(const Card& rhs) const;
};

class PokerHand : public Hand { . . . }
class BlackjackHand : public Hand { . . . }
```

get()

sort()

score()

~Hand()

get()

[2617] Examine the class hierarchy below. Assume that both derived classes are concrete and completely defined. Which of the following member functions must be overridden in the derived classes?

```
class Hand {
std::vector<Card> cards;
public:
Hand() = default;
virtual ~Hand() = default;
void add(const Card&);
virtual int score() const = 0;
virtual void sort();
bool operator<(const Card& rhs) const;
};

class PokerHand : public Hand { . . . }
class BlackjackHand : public Hand { . . . }
```

add()

get()

score()

sort()

score()

derived classes allowed (but not required to) override?

class Hand {
std::vector<Card> cards;
public:
Hand() = default;
virtual ~Hand() = default;
void add(const Card&);
virtual int score() const = 0;
virtual void sort();
bool operator<(const Card& rhs) const;
};

class PokerHand : public Hand { . . . }
class BlackjackHand : public Hand { . . . }

get()

score()

add()

sort()

In C++, as in Java pure virtual member functions may not have an implementation

The C++ facility that allows a derived class to have multiple base classes is known as interface inheritance

In C++, an Abstract Base Class is any class that has one or more virtual member functions

The istream class in the C++ standard library uses multiple inheritance

Since an abstract class cannot be instantiated, it is illegal to have references of abstract types

Using public inheritance to derive Stack from vector is a good design because vector provides all of the capabilities that a Stack requires

An abstract class is a class that contains no data members

Constructing an instance of an abstract class is legal, provided you do not initialize it

Consider the Shape class hierarchy, along with Circle, Square and Star from your text. The Shape class is a concrete class

What Java calls a static method is called a pure virtual member function in C++

If a class is abstract, you may create instances, but not pointers of that class

An abstract class may, but is not required to, override its pure virtual (abstract) member functions

Composition models an IS-A relationship between classes

Composition can be used to create adapter classes that change the implementation of one class to meet the needs of another

Private inheritance models an IS-A relationship between classes

In C++, public inheritance can be used to create adapter classes

An abstract class is a class that contains only virtual member functions

Using the keyword abstract to the heading of a virtual member function converts it to a pure virtual member function

Abstract classes provide a set of capabilities that derived classes my inherit

In adapter classes, the member functions override superclass member functions to provide new behavior

Consider the Shape class hierarchy, along with Circle, Square and Star from your text. The Circle class is an abstract class

---

False

It is illegal to construct an instance of an abstract class Abstract classes specify a set of responsibilities that derived classes must fulfill The iostream class in the C++ standard library uses multiple inheritance In C++, an Abstract Base Class is any class that has one pure virtual member function An abstract class is a class that contains member functions that are specified but not implemented The C++ facility that allows a derived class to have multiple base classes is known as multiple inheritance In C++ pure virtual member functions may have an optional implementation Adding = 0 to the end of the heading of a virtual member function converts it to a pure virtual member function An abstract class requires its concrete derived classes to override all of its pure virtual (abstract) member functions In composition-based adapter classes, the member functions delegate or forward requests to the data member that can satisfy the request In C++, private inheritance can be used to create adapter classes Public inheritance models an IS-A relationship between classes Using public inheritance to derive Stack from vector is a problem because a Stack is really not a vector If a class is abstract, you may create a pointer of that class What Java calls an abstract method is called a pure virtual member function in C++ Consider the Shape class hierarchy, along with Circle, Square and Star from your text. The Shape class is an abstract base class	
[1213] What is stored in data after this runs? vector<int> data{1, 2, 3}; data.back(); None of these [] [1, 2, 3, 0] [2, 3] [1, 2] [1, 2, 3]	[1,2,3]

[1540] What is printed?
template <typename T>
ostream& mystery(ostream& out, const T* p, size_t n)
{
out << '[';
if (n) {
out << p[0];
for (size_t i = 1; i < n; i++)
out << ", " << p[i];
}
out << "]";
return out;
}
. . .
int a[] = {1,2,3,4,5,1};
mystery(cout, a, sizeof(a) / sizeof(a[0])) << endl;
None of these or undefined output.
[1, 2, 3]
[1, 2, 3, 4, 5, 1]
[1, 2, 3, 4]
[1, 2, 3, 4, 5]
	[1, 2, 3, 4, 5, 1]

[1420] What is the equivalent array notation? int dates[10]; cout << (*dates) + 2 << endl; &dates[2] dates[0] + 4 dates[2] dates[0] + 2 dates[2] + 2	dates[0] + 2

[1822] What prints?
int cnt = 0, a[4][5];
for (int i = 0; i < 5; i++)
for (int j = 0; j < 4; j++)
a[j][i] = cnt++;
cout << a[3][2] << endl;
8
14
11
9
19
	11

int sum = 0;
for (auto e : a) sum =+ e;
cout << "sum->" << sum << endl;
sum->20
Does not compile. Cannot use range-loop on arrays.
sum->8
Compiles and runs, but results are undefined.
Does not compile; e is undefined.	

[1304] Assume that ppi correctly points to pi. Which line prints the address of ppi?
int main()
{
double pi = 3.14159;
double *ppi;
// code goes here
// code goes here
}
cout << &ppi;
None of these
cout << π
cout << ppi;
cout << *ppi;
	cout << &ppi;

[1811] Which statement displays the element appearing in the second row and the third column? cout << a[1][2]; cout << a[2][1]; cout << a[2][3]; None of these cout << a[3][2];	cout << a[1][2];

[1714] What happens here?
char s1[] = "CS150", s2[10];
strcpy(s1, s2);
s2[0] = 'X';
cout << s1 << endl;
"XS150"
Crashes when run.
"CS150"
Undefined behavior
Does not compile
	Undefined behavior

[1607] Below is a declaration for a partially-filled array. What is the correct prototype for a function delete() that deletes the element at position pos in the array, shifts the remaining elements left, and returns true if successful?
const size_t MAX = 100;
double nums[MAX];
size_t size = 0;
None of these
bool delete(double a[], size_t MAX, size_t& pos);
bool delete(double a[], size_t size, size_t pos);
bool delete(const double a[], size_t& size, size_t pos);
bool delete(double a[], size_t& size, size_t pos);
	bool delete(double a[], size_t& size, size_t pos);

[1923] This code:
int * f()
{
int a[] = {1, 2, 3};
return &a[1];
}
has a syntax error
None of these
has a memory leak
has a dangling pointer
has a double delete
	has a dangling pointer

[1721] Which lines create the C-string "hello"? 1. char s[10] = "hello"; 2. char s[10] = {'h','e','l','l','o'}; 3. char s[] = {'h','e','l','l','o','0'}; 4. char s[5] = "hello"; 5. char s[] = "hello"; 1, 2, 3, 5 1, 3 All of them 1, 2, 5 1, 5	1, 2, 5

[1536] What does this function do?
double mystery(const double a[], size_t len)
{
double x = 0;
for (size_t i = 0; i < len; i++)
if (a[i] < x) x = a[i];
return x;
}
Returns the smallest number in the array
Undefined. Depends on the input.
Does not compile
Returns the largest number in the array
	Undefined. Depends on the input.

Text Initialized Data Stack Heap	
[1803] What prints? Assume 4 bytes per int. int a[][2] = {0}; cout << sizeof(a) << endl; 4 Illegal declaration. Does not compile. 12 8 16	8
[1328] What is a common pointer error? Setting a pointer value to nullptr Assigning a new value to a pointer Dereferencing a pointer Using indirection on a pointer Using a pointer without first initializing it	Using a pointer without first initializing it
[1710] What happens here? char s[50] = "CS150"; strcat(s, "CS50"); cout << s << endl; "CS150CS50" Crashes when run. Undefined behavior "CS500" "CS50"	"CS150CS50"

[1415] Which returns the last pixel on the first row of this image?
Pixel *p; // address of pixel data
int w, h; // width and height of image
p + w - 1
None of these are correct
*(p + w) - 1
*(p + w - 1)
*p + w - 1
	*(p + w - 1)

[1606] Below is a declaration for a partially-filled array. What is the correct prototype for a function insert() that inserts a new element at position pos in the array, shifts the remaining elements right, and returns true if successful?
const size_t MAX = 100;
double nums[MAX];
size_t size = 0;
bool insert(double a[], size_t& size, double e, size_t pos);
None of these
bool insert(double a[], size_t& size, size_t MAX, double e, size_t pos);
bool insert(double a[], size_t MAX, double e, size_t pos);
bool insert(double a[], size_t size, size_t MAX, double e, size_t pos);
	bool insert(double a[], size_t& size, size_t MAX, double e, size_t pos);

[1906] The variable *p:
void f()
{
int *p = new int(42);
}
is uninitialized
stores the value 42 in all versions of C++
stores a memory address
stores the value 42 in C++11 only
is undefined. Code does not compile.
	stores the value 42 in all versions of C++

[1401] Which of these lines correctly prints 3?
struct S {
int a = 3;
double b = 2.5;
};
S obj, *p = &obj;
cout << p.a << endl;
cout << *p.a << endl;
cout << *(p.a) << endl;
cout << (*p).a << endl;
cout << *(p).a << endl;
	cout << (*p).a << endl;

{
vector<int> v{1, 2, 3};
auto size = v.size();
cout << v.back() << endl; // 1.
cout << v.front() << endl; // 2.
cout << v.at(0) << endl; // 3.
cout << v.at(size) << endl; // 4.
cout << v.pop_back() << endl; // 5.
}
1
4
5
2
3	

[1801] Which of these is a 2D array? int c[2][2]; int d[][] int a[][2]; All of these int *b[2];	int c[2][2];
[1323] What is true about an uninitialized pointer? None of these are true Dereferencing it is undefined behavior Dereferencing it will cause a program crash It is set to the nullptr value Dereferencing it is safe, but has no effect.	Dereferencing it is undefined behavior

[1516] What is printed here? (Assume all includes have been added. Assume 4-bytes per int, 8 bytes per pointer.)
size_t len(const int a[])
{
return sizeof(a) / sizeof(a[0]);
}
int main()
{
int a[] = {2, 4, 6, 8};
cout << len(a) << endl;
}
2
Does not compile
4
1
	2

[1601] Below is a partially-filled array. If you are adding elements to this array in a loop, what is the correct loop boundscondition?
const size_t MAX = 100;
double nums[MAX];
size_t size = 0;
for (size = 0; size < MAX; size++) . . .
while (MAX < size) . . .
while (size < MAX) . . .
while (size <= MAX) . . .
None of these
	while (size < MAX) . . .

[1922] This code:
void f()
{
int *p = new int[3]{rand(), rand(), rand()};
if (p[1] != 0 && p[2] != 0) delete[] p;
cout << p[0] / p[1] / p[2] << endl;
}
has a memory leak
None of these
has a dangling pointer
has a syntax error
has a double delete
	has a dangling pointer

[1214] What prints?
void f(const vector<int>& v)
{
v.at(0) = 42;
}
int main()
{
vector<int> x{1, 2, 3};
f(x);
cout << x.at(0) << endl;
}
Nothing; compile-time error.
1
42
Nothing; linker error
Nothing; run-time error.
	Nothing; compile-time error.

sizeof(countries) sizeof(countries) / sizeof(countries[0]) sizeof(countries) * sizeof(countries[0]) None of these len(countries)	

[1307] The value for the variable b is stored:
int a = 1;
void f(int b)
{
int c = 3;
static int d = 4;
}
on the heap
in the CPU machine registers
on the stack
in the static storage area
The example does not provide enough information
	on the stack

[1907] The variable *p:
void f()
{
int *p = new int{};
}
stores the value 0 in C++11 only
is uninitialized
stores the value 0 in all versions of C++
is undefined. Code does not compile.
stores a memory address
	stores the value 0 in C++11 only

[1506] Below is a cumulative algorithm using an array and an iterator-based loop. What is printed? (Assume all includes have been added, etc.)
double average(const int **beg, const int** end)
{
double sum = 0;
size_t count = end - beg;
while (beg != end) sum += *beg++;
return sum / count;
}
int main()
{
int a[] = {2, 4, 6, 8};
cout << average(begin(a), end(a) - 1) << endl;
}
Endless loop when run; likely crashes.
5
6
4
Does not compile
	4

[1701] Where are the characters "Hello" stored in memory?
char s1[1024] = "Hello";
void f()
{
const char *s2 = "Goodbye";
char s3[] = "CS 150";
}
static storage area (read-only)
stack
heap
static-storage area (read/write)
None of these
	static-storage area (read/write)

[1904] The variable p:
void f()
{
int *p = new int;
}
None of these
stores a memory address
stores the value 0
is uninitialized
	stores a memory address

Examine the following lines which build a utility library for manipulating digits. Match each term with the correct response: digit-tester: digits.o digit_tester.o clang++ digits.o digit_tester.o -o digit-tester Executable Object file Interface file Project file Client file Implementation file	Executable -> digit-tester Object file -> digits.o Interface file -> digits.h Project file -> makefile Client file -> digit_tester.cpp Implementation file -> digits.cpp
A tool named Doxygen is often used to generate HTML user docs from C++ code. Correct! True False	True

False	
Implementation files must explicitly qualify each name from the standard library with std:: True False	False
An undeclared error message is a compiler error. True False	True
In a library, the client or test program: consists of function definitions consists of instructions that produce the executable consists of declarations or prototypes consists of function calls None of these	Consists of function calls
Which of these are targets? EXE=digit-tester OBJS=client.o digits.o $(EXE): $(OBJS) $(CXX) $(CXXFLAGS) $(OBJS) -o $(EXE) client.o digit-tester digits.o $(EXE) None of these	Digit-tester $(EXE)
Below are several functions and member functions involved in text I/O. Match the function or method with the correct description. -Has a single char&parameter -Returns the last character read to the input stream -Examines, but does not read the next character in an input stream -Replaces the last character read with any character -Called implicitly when an input statement is used as a test condition. -A predicate function -Converts its value argument to a character and sends it to output.	get() unget() peek() putback() fail() isalpha() put()
Default arguments appear only in the function implementation. True False	False
Default arguments appear only in the function prototype. True False	True
Default arguments let you call a single function in several different ways. True False	True
Default arguments may only be used with reference parameters. True False	False

What prints?
void fn(int, double, double&) { cout << "A" << endl; }
void fn(int, int, double&) { cout << "B" << endl; }
void fn(int, int, double) { cout << "C" << endl; }
void fn(int, int, int) { cout << "D" << endl; }
int main()
{
fn(2.5, 1.5, 2.5);
}
C
A
D
Syntax error: no candidates
B
Syntax error: ambiguous
	C

Given the overloaded functions prototypes and the variable definition below, which of the function calls will fail to compile?
int f(int&);
int f(int);
int f(int, int);
int a = 7;
// Options:
f(3)
f(a);
None of these fail to compile
f(2.0);
f('a', 'b')
	f(a);

-different number of parameters -different return type -different parameter types -different order of parameter types. -different function name	different function name

Which line in the function "skeleton" below contains an error?
#include "borgia.h" // 1.
void primoTiara(int n) // 2.
{ // 3.
return 0; // 4.
} // 5.
None of these
// 3.
// 1.
// 2.
// 4.
// 5.
	//4.

Which prototype(s) in the following header file are syntactically correct (legal)?
#ifndef EXAMPLE_H
#define EXAMPLE_H
#include <string>
string f1(int a);
int f2(double);
void f3(std::string& s, int n);
double f4();
#endif
f4
f3
f1
f2
	f3
	f2

Examine this code. Which is the best prototype?
string s = "pig";
cout << latin(s) << endl; // igpay
cout << s << endl; // pig
string latin(string&)
string latin(string);
None of these
string latin(const string&)
void latin(string&)
	string latin(const string&)

If f() needs to change the argument passed here, the parameter must be declared as:
void f(. . . str);
int main()
{
string s = "hello";
f(s);
}
const string&
It is not possible for f() to change the argument passed here.
const string
string&
string
	string&

Which of these prototypes is the best one to use in this circumstance?
int main()
{
string str{"To be or not to be."};
cout << "Most common letter is "
<< mostCommon(str) << endl;
}
char mostCommon(const string&);
None of these are correct
char mostCommon(const string);
char mostCommon(string);
char mostCommon(string&);
Any of these are fine.
	char mostCommon(const string&);

Which of these are not ways that functions may be overloaded? -different parameter types -different function name -different return type -different parameter names -different order of parameter types. -different number of parameters	different function name different return type different parameter names

void f(. . . str);
int main()
{
f("hello");
}
-string
-It is not possible for f() to accept the argument passed here.
-const string
-const string&
-string&	

A process filter learns something about the stream by examining characters. True False	False
To test if an I/O operation succeeded you must explicitly call the stream's fail() member function. True False	False
A state filter learns something about the stream by examining characters. True False	True
Calling cout.put(65) is illegal. Your code will not compile. True False	False

What does this filter do?
char ch;
while (cin.get(ch))
{
if (isspace(ch) && isspace(cin.peek()))
continue;
cout.put(ch);
}
Compresses spaces in a line and single-spaces lines of input
None of these
Single spaces input lines only
Compresses spaces to a single space only
	Compresses spaces in a line and single-spaces lines of input

What does this filter do?
char ch;
int x = 0;
while (cin.get(ch))
{
if (ch == '\n') x++;
}
cout << x << endl;
-Counts the number of lines in input
-Counts the number of characters in input.
-Counts the number of non-blank lines in input
-None of these
	Counts the number of lines in input

What does this filter do?
char ch;
while (cin.get(ch))
{
if (isdigit(ch)) cout.put('0');
}
-Replaces all non-digits with '0'
-Replaces all digits in a file with '0'
-None of these
-Replaces only the first digit in a file with '0'.
	Replaces all digits in a file with '0'

What does this filter do?
char ch;
while (cin.get(ch))
{
if (isspace(ch)) break;
cout.put(ch);
}
-Removes all spaces from input; prints each line separately
-None of these
-Removes all spaces from input; prints a single line of output
-Prints only first word; stops on first space
	Prints only first word; stops on first space

What kind of error is this? ~/workspace/ $ ./ex1 The Patriots won the 2018 Super Bowl -Runtime error (throws exception when running) -Compiler error (something is missing when compiling) -Linker error (something is missing when linking) -Type error (wrong initialization or assignment) -Syntax error (mistake in grammar) -None of these -Operating system signal or trap	None of these

a = "hello world'; ^ ex1.cpp:7:9: error: expected expression -Operating system signal or trap -Syntax error (mistake in grammar) -Runtime error (throws exception when running) -Linker error (something is missing when linking) -Type error (wrong initialization or assignment) -None of these -Compiler error (something is missing when compiling)	
Calling a template function like to_string(3.5) is known as implicit instantiation. True False	True
A template function may be declared in a header file but must be defined in an implementation file. True False	False
Calling a template function like to_string<int>(3.5) is known as implicit instantiation. True False	False
When you throw an exception, control immediately jumps out of the current try block. True False	True
Which of the following statements throws a valid exception in C++? -4 throw; -throw.function(); -throws str; -throw 2;	throw 2;

Complete the code fragment below, which is designed to throw an illegal_length exception if string variable accountNumber has more than seven characters.
if (accountNumber.size() > 7)
{
_____;
}
-throws illegal_length("Account number exceeds maximum length");
-throws new illegal_length("Account number exceeds maximum length");
-throw illegal_length("Account number exceeds maximum length");
-throw new illegal_length("Account number exceeds maximum length");
	throw illegal_length("Account number exceeds maximum length");

What happens when this code fragment runs in C++ 11? cout << sqrt(-2) << endl; - -1.41421 is printed -It throws a runtime exception -sqrt() returns a not-a-number error value -It sets an error state in cout. -None of these -It does not compile.	sqrt() returns a not-a-number error value

What prints?
string s("hello");
try {
if (s.size() > 20) throw 42;
if (isupper(s.back())) throw "goodbye";
if (s == "Hello") throw string("hello");
s[s.size()] = 'x';
cout << "one\n";
}
catch (const int& e) { cout << "two\n"; }
catch (const string& e) { cout << "three\n"; }
catch (exception& e) { cout << "four\n"; }
catch (...) { cout << "five\n"; }
-five
-one
-Undefined (print one or crash)
-four
-two
-three
	Undefined (print one or crash)

try {
if (s.size() > 5) throw s.size();
if (isupper(s.back())) throw s.back();
if (s == "hello") throw string("hello");
s.at(s.size()) = 'x';
cout << "one\n";
}
catch (const string& e) { cout << "two\n"; }
catch (exception& e) { cout << "three\n"; }
catch (...) { cout << "four\n"; }
-four
-three
-two
-Undefined
-One	

Formatted I/O means that you read and write data line-by-line. True False	False
A loop that reads data until some special value is found is called a data loop. True False	False
If an input stream's file is missing when you try to open it, its fail() member function returns true. True False	True
In the flag-controlled-pattern, you use a break statement to exit the loop when the sentinel is found. True False	False
Stream arguments to a function should always be passed: -by const reference -by reference -by value -by reference for input, and const reference for output -None of these	By reference

What does this code do?
ifstream in("temp.txt");
char x;
int i{0};
while (in.get(x)) i++;
cout << i << endl;
-Counts the number of non-space characters in the file
-Counts the number of words in the file
-Gets stuck in an endless loop
-Counts the number of lines in the file
-Counts the number of digits in the file
-Counts the number of characters in the file
	Counts the number of characters in the file

The file temp.txt contains "Orange Coast College". What prints?
ifstream in("temp.txt");
char c;
while (in.get(c))
{
if (isupper(c))
cout << toupper(c);
}
-occ
-ORANGE COAST COLLEGE
-oRANGE cOAST cOLLEGE
-OCC
-range oast ollege
	OCC

Which line reads a single word from the istream named in into the string variable word? -in.get(word); -None of these -getline(in, word); -word = in.next(); -in << word;	none of these

Which call below produces 5?
template <typename T>
void addem(T a, T b)
{
cout << a << " + " << b << "->"
<< (a + b) << endl;
}
-addem<int>(3, 2.5);
-addem(3, 2.5);
-None of these
-addem(3.0, 2.5)
-addem<double>(3, 2.5);
	addem<int>(3, 2.5);

False	
User-defined types that combine multiple values into a single type are called scalar types. True False	False
When passing a structure variable to a function, use non-const reference if the function should not modify the actual argument. True False	False
In C++, objects have value semantics; object variables contain the data members. True False	True
Examine the following code (which is legal). Which statement is illegal (given only this code)? struct Money { int dollars{0}, cents{0}; } m1, m2; -m1 = m2; -if (m1.cents != m2.dollars) . . . -m2.cents++; -cout << m1 << endl;	cout << m1 << endl;

Given the following structure and variable definitions, which data members are initialized?
struct Employee
{
long empID;
std::string lastName;
double salary;
int age;
};
Employee bob;
-salary
-age
-None of these
-lastName
-empID
	lastName

Given the following structure and variable definitions, which data members are default initialized?
struct Employee
{
long empID;
std::string lastName;
double salary;
int age;
};
Employee bob{777, "Zimmerman", 5000000.0, 76};
-None of these
-salary
-age
-lastName
-empID
	None of these

Given the following structure and variable definitions, which data members are uninitialized?
struct Employee
{
long empID;
std::string lastName;
double salary;
int age;
};
Employee bob;
-None of these
-lastName
-empID
-salary
-age
	empID
	salary
	age

Match each item with the correct loop form below. (1)Limit loop that reduces its input - (2)Limit loop that uses successive approximations - (3)Counter-controlled symmetric loop for producing a sequence of data - (4)Data loop that uses raw input - (5)Counter-controlled asymmetric loop for processing characters - (6)Iterator loop that may change elements in its container - (7)Iterator loop that cannot change elements in its container - (8)Counter-controlled loop for processing substrings - (9)Data loop that uses formatted input	(1)While(n != 0){n /= 2;} - (2)While(abs(g1-g2) >= EPSILON {...} - (3)For(int I = 12; I <= 19; i++) {...} - (4)While(cin.get(ch)){...} - (5)For(size_t I = 0, len = s.size(); I < len; i++) {...} - (6)for (auto& e : col) {. . .} - (7)for (auto e : col) {. . .} - (8)for(size_t I = 4, slen = 4, len = s.size(); I <= len; i++) {...} - (9)while(cin >> n) {. . .}

for (auto e : s)
{
if (toupper(e))
out.put('x');
}
-inline test
-sentinel loop
-iterator or range loop
-data loop
-limit loop
-primed loop
-loop-and-a-half
-counter-controlled loop

Which of the following loop patterns are used here?
string s{"hello CS 150"};
for (auto e : s)
{
if (toupper(e)) break;
}
-loop-and-a-half
-iterator or range loop
-inline test
-sentinel loop
-data loop
-primed loop
-limit loop
-counter-controlled loop

loop-and-a-half

Which of these statements apply to C++?
-Low-level language
-More efficient than Java or Python.
-Interpreted by a virtual machine
-Produces native code that runs on the CPU
-Automatically catches errors like array out of bounds.
-Compiles to native code
-Compiles to bytecode

More efficient than Java or Python.
Produces native code that runs on the CPU
Compiles to native code

[0309] How is your nesting instinct? What prints? (Carefully check each operator and semicolon.)
#include <iostream>
using namespace std;
int main()
{
int x = 4;
if (x <= 2);
if (x == 4) {
cout << "one" << endl;
}
else cout << "two" << endl;
}
-one
-Nothing; does not compile
-onetwo
-Compiles, runs, but prints nothing.
-two

One

[0503] Examine the loop plan from your reader below. Line # 9 (underlined):
1. Goal: count the characters in a sentence ending in a period
2. Given: the variable str is a string (may be empty)
3. Let counter = -1
4. If str has any characters Then
5. Let counter = 0
6. Let current = first character in str
7. While str has more characters and current is not a period
8. Add 1 to counter
9. Let current = next character in str
10. If current is a period Then Add 1 to counter
11. Else Let counter = -2
12. If counter is -1 Then the string str is empty
13. ElseIf counter is -2 Then no period was found in str
14. Else counter contains the count
-is the loop bounds
-is a goal precondition
-None of these
-is the goal operation
-advances the loop

Advances the loop

[0437] Assume a is 20 and b is 21; what prints?
// 0123456789'123456789'123
string s = "The elephant in the room";
cout << s.substr(a, b) << endl;
-"r"
-Runtime error
-"room"
-"room

"room"

-necessary bounds -intentional bounds -None of these -symmetric bound -asymmetric bounds	
[0305] What manipulator can you use to ensure that his large floating-point number appear using regular decimal notation? -fixed -decimal -setw -setprecision -scientific -hex	fixed
[0525] Look at the problem statement below. The _____ of the loop is that a period was encountered. How many characters are in a sentence? Count the characters in a string until a period is encountered. If the string contains any characters, then it will contain a period. Count the period as well. goal -None of these -bounds -plan	bounds
[0502] Examine the loop plan from your reader below. Line # 5 (underlined): 1. Goal: count the characters in a sentence ending in a period 2. Given: the variable str is a string (may be empty) 3. Let counter = -1 4. If str has any characters Then 5. Let counter = 0 6. Let current = first character in str 7. While str has more characters and current is not a period 8. Add 1 to counter 9. Let current = next character in str 10. If current is a period Then Add 1 to counter 11. Else Let counter = -2 12. If counter is -1 Then the string str is empty 13. ElseIf counter is -2 Then no period was found in str 14. Else counter contains the count -is a bounds precondition -None of these -is a goal precondition -is the loop bounds -advances the loop	-is a goal precondition

[0422] This compiles, runs and prints 12. What is the correct parameter declaration for x?
int x = 6;
multiply(x, 2);
cout << x << endl;
-int x
-None of these
-int& x
-const int& x
	int& x

[0441] Which line throws an exception because of range checking? 1. string s = "holey moley"; 2. auto len = s.size(); 3. auto a = s.front(); 4. s.at(len) = a; 5. s[len] = 'c'; -2 -5 -3 -4 -None of these	4

[0221] Which of these lines is illegal?
#include <iostream>
using namespace std;
/**1**/ int a, b;
/**2**/ a = 3;
int main()
{
/**3**/ b = 4;
/**4**/ cout << a << ", " << b << endl;
}
-None of these lines
-2
-1
-4
-3
	2

-if -switch -if . . . else . . . if . . . else -if . . . else -if . . . if . . . else . . . else	

[0424] How many variables appear in the following code segment?
int n = 5;
int& r1 = n;
auto& r2 = r1;
r1 = 4;
r2 = 3;
cout << n << endl;
-1
-2
-None of these
-3
	1

[0501] Examine the loop plan from your reader below. Line # 6 (underlined): 1. Goal: count the characters in a sentence ending in a period 2. Given: the variable str is a string (may be empty) 3. Let counter = -1 4. If str has any characters Then 5. Let counter = 0 6. Let current = first character in str 7. While str has more characters and current is not a period 8. Add 1 to counter 9. Let current = next character in str 10. If current is a period Then Add 1 to counter 11. Else Let counter = -2 12. If counter is -1 Then the string str is empty 13. ElseIf counter is -2 Then no period was found in str 14. Else counter contains the count -None of these -is a bounds precondition -is the loop bounds -is a goal precondition -advances the loop	is a bounds precondition
[0530] In the classic for loop, which portion of code is not followed by a semicolon? -condition expression -update expression -None of these -initialization statement	update expression
[0506] Examine the loop plan from your reader below. Line # 4 is 1. Goal: count the characters in a sentence ending in a period 2. Given: the variable str is a string (may be empty) 3. Let counter = -1 4. If str has any characters Then 5. Let counter = 0 6. Let current = first character in str 7. While str has more characters and current is not a period 8. Add 1 to counter 9. Let current = next character in str 10. If current is a period Then Add 1 to counter 11. Else Let counter = -2 12. If counter is -1 Then the string str is empty 13. ElseIf counter is -2 Then no period was found in str 14. Else counter contains the count -a loop guard -a necessary condition -an intentional condition -None of these -a boundary condition	a loop guard
To the right are the six steps of the C++ development process as shown in your textbook. Below, match each step with the tool that accepts the shown input and produces the output as shown. Step 1 Step 2 Step 3 Step 4 Step 5 Step 6	(1) Text editor (2) Preprocessor (3) Compiler (4) Linker (5) Loader (6) CPU

(1) Function call (2) Function prototype (3) Function definition (4) Prompt (5) Variable definition (6) Input statement (7) Expression (8) Parameter (9) Output statement (10) Program entry point (11) Documentation comment (12) Standard library headers (13) Namespace directive (14) Optional return	(3) #12 (4) #6 (5) #7 (6) #8 (7) #13 (8) #14 (9) #10 (10) #5 (11) #1 (12) #2 (13) #3 (14) #11
Assume that the user enters: Jimmy Paz 68 3.5 What value is stored in age? mc012-1.jpg undefined .5 3.5 3	Undefined
Assume that name is a string object. Which of these expressions are legal? -name += 'X' -name += "fred" -name = "sally" + name -name < "bob" -name == "sally" -name = name + 777 -name.equals("bob") -"sally" += name	Name += 'X' Name += "fred" Name = "sally" + name Name < "bob" Name == "sally"
Assume c is a char variable. Which line produces a syntax error? mc011-1.jpg You Answered -5 -2 -4 -None of these -3	None of these
Which of these selects a character (char) from a string? -auto c = s.substr(0, 1); -auto b = s.charAt(0); -auto a = s[0]; -None of these	Auto a = s[0];
What value is stored in a after this runs? mc022-1.jpg - -1 -string::npos -None of these -7	String::npos
What value is stored in a after this runs? -"ABC" -None of these -"ABCD" -"BCDE" -"BCD"	"BCDE"
Match each item with the correct statement below. (1) Associates a name with a type (2) Read a value and store it in a variable (3) Copy a new value into an existing variable (4) Allocates space for a variable (5) Provides a starting value when a variable is created (6) A named storage area that holds a value	(1) Declare (2) Input (3) Assign (4) Define (5) Initialize (6) Variable
Match each item with the correct statement below. Assume int x, y, z; (1) Shorthand assignment (2) Post increment (3) Undefined behavior (4) Widening conversion (5) Pre decrement (6) Chained assignment (7) Narrowing conversion (8) Mixed-type expression	(1) Y += z; (2) X++; (3) X = z++ - ++z; (4) Double a = y; (5) -z; (6) X = y = z = 10; (7) Z = 3.15 (8) Auto v = x* 2.3;
Which operator is used to see if any of a set of conditions is true? -logical and -none of these -logical or -conditional operator -logical not -equality	logical or

(2) Handle an on or off condition such as a light switch (3) Handle numeric selections made from a menu (4) Process a group of radio buttons (5) Process income taxes for different incomes and filing statuses (6) Set a variable to one of two possible values	(3) The switch statement (4) Sequential if statements (5) Nested if statements (6) The conditional operator

[0325] What prints when you enter 2?
int which;
cin >> which;
cout << "The elephant in the room ";
if (which % 2)
cout << "is white!" << endl;
else
cout << "is pink!" << endl;
-None of these
-The elephant in the room is white!
-The elephant in the room is pink!
-The elephant in the room is pink!
-The elephant in the room is white!
	The elephant in the room is pink!

[0312] What happens here? (Carefully check each operator and semicolon.)
int y = 4;
if (y < 0);
y = -y;
cout << y << endl;
- -4
- Runtime error
-Syntax error
- Output is undefined.
-4
	-4 (negative)

[0301] How do you call the function shown here?
string square(int a)
{
return to_string(a * a);
}
-int a = square(4);
-None of these are legal.
-string a = square(42);
-All of these are legal.
-double a = square(4.0);
-string a; a.square(3);
	string a = square(42);

What is printed when this runs?
#include <iostream>
Using namespace std;
Int main()
{
Int a = 3;
Int b = ++a - a++;
Cout << "b->" << b << endl;
}
-0
-Anything at all because this operation is undefined.
-2
-1
	Anything at all because this operation is undefined.

What is printed when this runs?
#include <iostream>
Using namespace std;
Int main()
{
Int a = 3, b = ++a;
Cout << "a->" << a << ", b->" << b << endl;
}
-a->4, b->3
-This is a syntax error.
-a->4, b->4
-Anything, this is undefined beahvior.
	a->4, b->4

Before you run your program, asking the operating system to connect standard output to a file is called redirection.	True
When more than one match is found for the proffered arguments.	ambiguity
A function where an argument is converted to match a parameter	best match
A function where each argument is the same type as the corresponding parameter.	exact matches
A group of functions with the same name.	candidate set
A group of functions that have the same name and the correct number of parameters. When no match is found for the proffered arguments	viable set setempty set
Explicitly initializing an array like this: int a[3] = {1, 2, 3}; requires the size and the number of elements supplied to be the same.	False