Midterm 3

Share

16 studiers today ★ Leave the first rating

Terms in this set (325)

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>&); are correct
What is stored in data after this runs? vector<int> data{1, 2, 3}; data.front();	[1,2,3]

In C++ the parameterized collection classes are called _____?	templates

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

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 OR cout << v.back() << endl;

Assume vector<double> speed(5);
Which line throws a runtime error?
	NONE OF THESE:
	
	speed[0] = speed.back();
	speed.erase(speed.begin());
	cout << speed[5];
	speed.front() = 12;

What is the size of data, after this runs? vector<int> data; data.push_back(3);	1

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

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
	
	False:
	Code will not compile
	Prints 3
	Code compiles but gives a warning
	Prints 0

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

The declaration: vector<int> v = new vector<>(); creates a vector object with no elements.	False

vector subscripts begin at 1 and go up to the vector size.	False
Assume vector<int> v; Writing cout << v.front(); throws a runtime exception.	False
Elements in a vector are accessed using a subscript.	True
Assume the vector v contains [1, 2, 3]. v.pop_back(); changes v to [1, 2].	True
The declaration: vector<string> v{"bill", "bob", "sally"}; creates a vector containing three string objects.	True
The declaration: vector<int> v(10); creates a vector object containing ten elements initialized to 0.	True
Assume the vector v contains [1, 2, 3]. v.erase(v.begin() + 2); changes v to [1, 2].	True
The clear() member function removes all the elements from a vector.	True
Assume the vector v contains [1, 2, 3]. v.erase(0); changes v to [2, 3].	False
The statement v.insert(v.begin(), 3) inserts the element 3 into the vector v, overwriting the exiting element at index 0.	False

The statement v.insert(v.end() + 1, 3) is undefined because end() + 1 points past the last element in the vector.	True
A vector represents a linear homogeneous collection of data.	True
In the declaration: vector<int> v; the word vector represents the object's base type.	False
The declaration: vector<int> v; creates a vector variable but no vector object.	False
Assume that v contains [1, 2, 3]. The result of writing cout << v.at(4); is a compiler error.	False
Assume the vector v contains [1, 2, 3]. v.erase(0); is a syntax error. Correct!	True

[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

[1338] What is the term used to describe a variable with stores a memory address?	pointer

Midterm 3

Study

Match each item with the correct term below.
 	
Used to access the data inside a variable pointer
 	
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 type
	
	variable value
	
	pointer
	
	p = a;
	
	int x = 3;
	
	y = *a;
	
	double * v;
	
	sizeof(Star)
	
	nullptr

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

[1341] Which area of memory are local variables stored in?	Stack
[1322] In C++, global variables are stored:	in the static storage area

[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++;
	
	legal:
	p1++;
	p2++;
	*p1 = 3;
	*p3 = 7;

[1342] Which area of memory are global variables stored in?	Static storage area
[1321] All of these are legal C++ statements; which of them uses indirection? int a = 3, b = 4;	int x = *p;
[1332] What is printed when you run this code? int *n{nullptr}; cout << n << endl;	The address value 0

[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

[1320] All of these are legal C++ statements; which of them uses the C++ dereferencing operator? int a = 3, b = 4;	int x = *p;
[1335] What is printed when you run this code? int *p = &0; cout << *p << endl;	No output; compiler error
[1328] What is a common pointer error?	Using a pointer without first initializing it

int a = 3, b = 4;	
[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++;

[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

[1347] 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);

[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;
[1324] What is true about this code? int n{500}; int *p = &n;	*p is the value of n

[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 - 1)

[1425] What prints?
 	
int a[] = {1, 3, 5, 7, 9};
int *p = a;
cout << *++p;
cout << *p << endl;
	33

In C++ using == to compare one array to another is permitted (if meaningless).	True
The elements of a C++ array created outside of a function are allocated in the static-storage area.	True
[1421] What is the equivalent array notation? int dates[10]; cout << *dates + 2 << endl;	dates[0] + 2

[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;

The subscripts of a C++ array range from 0 to the array size - 1.	True
The reinterpret_cast instruction changes way that a pointer's indirect value is interpreted.	True
The static_cast instruction changes way that a pointer's indirect value is interpreted.	False

[1406] Which line throws and out_of_range exception? double speed[5] = {. . .}; cout << speed[1] << endl; cout << speed[4] << endl; cout << speed[0] << endl; cout << speed[5] << endl;	
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.	True
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);	True

[1426] What prints?
 	
int a[] = {1, 3, 5, 7, 9};
int *p = a;
cout << ++*p;
cout << *p << endl;
	22

In C++ assigning one array to another is illegal.	True

[1424] What prints?
 	
int a[] = {1, 3, 5, 7, 9};
int *p = a;
cout << *p++;
cout << *p << endl;
	13

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.	True
[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};	a2
The elements of a C++ int array with no explicit initialization, created in a function will be set to zero.	False
The size of the array is not stored along with its elements.	True
C++ arrays offer built-in member functions for inserting and deleting.	False

[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

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

[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 input

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;
}	

[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& size, size_t MAX, double e);

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

[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;
}
	4

In a partially-filled array, the size represents the allocated size of the array.	False
An array passed to a function is passed by reference.	False
An array passed to a function is passed by address.	True
When deleting an element from a partially-filled array, it is an error if the index of the element to be removed is < size.	False
The library function begin(a) returns a pointer to the element right before the first in the array a.	False
If p points to the first element in [1, 3, 5] then cout << *++p prints 3.	True
Arrays generally have higher performance than a vector.	True

The function mystery(const int**, const int**) likely employs a counter-controlled loop.	False
The parameter declarations int p* and int[] p mean the same thing.	False

[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;
}
	Return the smallest number in the array

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

[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, size_t MAX, double e, size_t pos);

[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;
	...

[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;
	14

[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;
	an address

[1725] Which library function performs an equivalent operation on C-strings? string s1 = "Hello"; string s2 = "World"; s1 = s1 + s2;	strcat()
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
[1804] What prints? Assume 4 bytes per int. int a[][2] = {{0},{0}}; cout << sizeof(a) << endl;	16
C-string assignment uses the strcpy() function.	True
[1812] What prints when this runs? int a[2][3] = {1, 2, 3, 4, 5, 6}; cout << a[0][2] + a[1][2] << endl;	9

[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;
	6

strcmp(s1, s2) returns a negative number if s1 is lexicographically "less than" s2.	True
The effective size of the C-string char * s1 = "hello"; is 5 characters, but 6 characters are used for storage.	True
The sizeof operator returns the effective size of a C-string allocated as an array.	False

[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

C-strings are char pointers to the first character in a sequence of characters, terminated with a '\0' character.	False
On the command line, argv is an array of C-style strings.	True
Arrays can have more than two dimensions in C++.	True

[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;
	9

The allocated size for the C-string char s1[] = "hello"; is 6 characters, while the effective size is 5 characters.	True
In a 2D array the first subscript represents the rows and the second the columns.	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
The statement new int; allocates an uninitialized integer on the heap.	True
Memory for global variables is allocated when the program is loaded from disk. This is known as dynamic allocation.	False
Requesting a block of memory from the operating system as the program runs is known as dynamic allocation.	True
To allocate memory on the stack, C++ uses the new operator.	False
Requesting a block of memory from the operating system as the program runs is known as automatic allocation.	False
Freeing unused memory that was allocated elsewhere in your program is done in C++ using manual memory management.	True
Requesting a block of memory from the operating system as the program runs is known as static allocation.	False
Memory for local variables is allocated on the stack when their definitions are encountered during runtime. This is known as automatic allocation.	True

Examine the following code and then answer the questions:
 	
struct X {
~X() { delete [] _d_; }
};
double a{42.5};
void f(double b) {
static double c = 3.14;
auto d = X{new double[3]{a,b,c}};
}
 	
The duration of the variable a is:
 	
The linkage of the variable c is:
 	
The scope of the variable b is:
 	
The duration of the variable d is:
	other incorrect Match Options:
	file
	internal
	programmer defined
	external
	
	static
	
	none
	
	block
	
	automatic

int * f()
{
int a[] = {1, 2, 3};
return &a[1];
}	

Given this declaration, which line below is illegal? auto p1 = unique_ptr<int>(new int{42});	auto p2 = p1;

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;
}
	424243

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;
}
	424243

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

The variable p points to:
void f()
{
int *p = new int[42];
}
	the first element of an array of 42 uninitialized ints

The variable p points to:
 	
void f()
{
int *p = new int[42]();
}
	the first element of an array of 42 ints with the value 0

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 memory leak

Examine this code. What goes on the blank line?
 	
void f()
{
int *p = new int{3};
. . .
_____
}
	delete p;

Examine this code. What goes on the blank line?
 	
void f()
{
int *p = new int{3};
. . .
_____
}
	delete[] p;

{
int *p = new int{3};
. . .
_____
}
Correct!
	delete *p;
	delete p[];
	delete p[3];
	delete[] p;

Which line correctly creates a smart pointer that points to the variable x? int x = 42;	NONE OF THESE unique_ptr<int>(&x); make_shared<int>(x); shared_ptr<int>(&x); unique_ptr<int[]>(&x);

Examine this code. What goes on the blank line?
 	
void f()
{
int *p = new int[3]{1, 2, 3};
. . .
_____
}
	NONE OF THESE
	
	delete p;
	delete p[];
	delete p[3];
	delete[] *p;

The variable *p:
 	
void f()
{
int *p = new int(42);
}
Correct!
	stores the value 42 in all versions of C++

C++ arrays can be allocated with a size of 0.	False
The elements of a C++ array created in a function are allocated on the heap.	False
[1419] What is the equivalent array notation? int dates[10]; cout << *(dates + 2) << endl;	dates[2]
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	True
[1429] Which expression returns the number of countries? string countries[] = {"Andorra", "Albania", . . . };	sizeof(countries) / sizeof(string)
[1405] What is stored in the last element of nums? int nums[3] = {1, 2};	0
A forward reference can be used when you want to use a structure as a data member without first defining the entire structure.	False
[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; Group of answer choices	**(p + 1)** 2
Group of answer choices	
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	False
C++ arrays have no support for bound-checking.	True
[1430] Which expression returns the number of countries? string countries[] = {"Andorra", "Albania", . . . };	sizeof(countries) / sizeof(countries[0])
[1411] Which assigns a value to the first position in letters? char letters[26];	letters[0] = 'a';
In C++ using == to compare one array to another is illegal.	False

[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;
	sums the elements in a

In C++ printing an array name prints the address of the first element in the array.	True
The reinterpret_cast instruction is allowed any time you want to change the type of a pointer.	False

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;	

The strcat() function cannot overflow the storage allocated for the destination buffer.	False

[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;
	11

The length of a C-string is never stored explicitly.	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
Given the C-string char * s3 = "hello"; strlen(s3) returns 6.	False
[1805] What prints? Assume 4 bytes per int. int a[][2] = {1, 2, 3}; cout << sizeof(a) << endl;	16
The C-string type is part of the standard library, not built into the C++ language.	False
[1710] What happens here? char s[50] = "CS150"; strcat(s, "CS50"); cout << s << endl;	"CS150CS50"

[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

When initializing a 2D, each column must have its own set of braces.	False
You can compare two C-strings, s1 and s2, by using the == operator.	False

[1717] What prints here?
 	
const char **a = "dog",** b = a;
if (a == b) cout << "dog == dog" << endl;
else cout << "dog != dog" << endl;
	dog == dog

The strncpy() function always appends a trailing NUL when the copy is finished.	False
The strncpy() function can be used to make sure that you don't copy more characters than necessary.	True
[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, 5
An abstract class is a class that contains only virtual member functions.	False
The istream class in the C++ standard library uses multiple inheritance	False
The public inheritance relationship is informally known as has-a	False
It is always a logic error for a derived class to redefine a non-virtual function	True

std::vector<Card> cards;
public:
Hand() = default;
virtual ~Hand() = default;

void add(const Card&);
Card get(size_t index) const;
virtual int score() const = 0;
virtual void sort();
};

class PokerHand : public Hand { .. }
class BlackjackHand : public Hand { .. }

As you know, C++ sometimes uses different terminology from Java and other Object-Oriented languages. According to this design, and using both C++ and traditional terminology, the PokerHand and BlackjackHand classes are described as ___

C++ term for child classes: derived

traditional OO: subclass

BOTH refer to classes w/ same parents as siblings

Here is a class hierarchy for different card games.
class Hand {
std::vector<Card> cards;
public:
Hand() = default;
virtual ~Hand() = default;
void add(const Card&);
Card get(size_t index) const;
virtual int score() const = 0;
virtual void sort();
};
class PokerHand : public Hand { .. }
class BlackjackHand : public Hand { .. }

As you know, C++ sometimes uses different terminology from Java and other Object-Oriented languages. According to this design, and using both C++ and traditional terminology, the Hand class may be described as a _____. There may be more than once correct answer.

abstract class & superclass

Base class w/ a single pure virtual fcn: abstract base class

OO term: super class

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 {
. . .
};

addSkill(), skills()

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 {.};

Window* p = new Container;

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 Hand::score() function because score() is not virtual

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 { . . . }	

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;
}
	not sure

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 = {Shape(), Square(), Oval()};
for (auto& e : v) iam(e);
cout << endl;
}
	not sure

The composition relationship is informally known as has-a	True
It is illegal to construct an instance of an abstract class	True
Abstract classes specify a set of responsibilities that derived classes must fulfill.	True
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.	True

Here is a class hierarchy for different card games.
 	
class Hand {
std::vector<Card> cards;
public:
Hand() = default;
virtual ~Hand() = default;
void add(const Card&);
Card get(size_t index) const;
virtual int score() const = 0;
virtual void sort();
};
 	
class PokerHand : public Hand { .. }
class BlackjackHand : public Hand { .. }
 	
According to this design, both the PokerHand and BlackjackHand may, but are not required to override which methods? Several answers may be correct.
	sort()
	
	not:
	add(), score(), get()
	
	regular virtual fcns may be overridden while pure-virtual fcns MUST be

class Hand {
std::vector<Card> cards;
public:
Hand() = default;
virtual ~Hand() = default;
void add(const Card&);
Card get(size_t index) const;
virtual int score() const = 0;
virtual void sort();
};
 	
class PokerHand : public Hand { .. }
class BlackjackHand : public Hand { .. }
 	
Both the PokerHand and BlackjackHand classes must override which member functions? Several answers may be correct.
	not: get(), add(), sort()
	
	only pure-virtual MUST be overridden

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;
Ducky duck;
	All of these will compile:
	
	pet = pup;
	Puppy& pr = pup;
	Pet* p = &duck;
	pet = kit;

What does this code mean?
 	
class X {
double x = Y().balance();
. . .
};
	Each X object uses- a Y object

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

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.	override

What does this code mean?
class X {
Y y;
. . .
};
	Every X object has-a Y object

Below is a class hierarchy. Which statements may result in slicing? 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();
A loop that reads data until the input stream signals that it is done is called a data loop.	T
Calling cout.put(65) prints the character 'A' on output.	T
Counting the number of words in input by counting word transitions is an example of a state filter.	T
The getline() function is a non-member function in the string library.	T

Midterm 3

Study

Match each item with the correct statement below. 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	unget() peek() putback() fail() isalpha() put()

What does this filter do?
 	
char ch;
while (cin.get(ch))
{
if (isspace(ch)) break;
cout.put(ch);
}
	Prints only first word; stops on spaces

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

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

Which line reads a single word from the istream named in into the string variable word?	None of these

What does this filter do?
 	
char ch;
while (cin.get(ch))
{
if (ispunct(ch)) continue;
cout.put(ch);
}
	Prints all non-punctuation characters in input

Stream arguments to a function should:	be as general as possible (istream & ostream)

What does this filter do?
char ch;
while (cin.get(ch))
{
if (isspace(ch)) continue;
cout.put(ch);
}
	unsure

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;
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). 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==

Midterm 3

Study

The structure and variable definitions are fine. Which statements are legal? struct Rectangle { int length, width; } big, little;	illegal: double p = 2 * (length + width); None of these are correct cin >> big; cout << Rectangle.length;

Examine the following code (which is legal). Which statement is legal?
 	
struct Money { int dollars{0}, cents{0}; } m1, m2;
	if (m1.dollars > m2.cents) ...
	
	illegal: m1 = {3, 4};
	cout << m1 << endl;
	if (m1 != m2) ...

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

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 statements are illegal?
 	
struct Money
{
int dollars{0};
int cents{1};
};
Money payment;
	payment{1} = 5;
	cout << Money.dollars;
	Money{1} = Money{0};

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;

The following code is illegal. struct {int hours, seconds; } MIDNIGHT{0, 0};	F
Structure variables should be passed to functions by value.	F
In C++, a collection of variables that have distinct names and types is called a structure.	F
Structures data members must all be of the same type.	F
Structures are homogenous data types.	F
C++ has two ways to represent records, the class and the struct.	T
The following code is legal. struct {int hours, seconds; } MIDNIGHT{0, 0};	T
You may create a structure variable as part of a structure definition.	T
In C++, a collection of variables that have distinct names and types is called a record.	F
When passing a structure variable to a function, use non-const reference if the intent is to modify the actual argument	T
This is the correct syntax for a C++ scoped enumeration. enum class WEEKEND {SATURDAY, SUNDAY};	T
Structures are heterogeneous data types.	T

The standard library types such as string and vector are scalar data types.	F
User-defined types that combine multiple values into a single type are called structured types.	T
User-defined scalar types are created with the enum class keywords in C++.	T
The built-in primitive data types such as int, char and double are scalar data types.	T
User-defined types that contain a single value are called structured types.	F
When passing a structure variable to a function, use const reference if the function should not modify the actual argument.	T
A structure member may be a variable of a different structure type	T

Examine the code below and match the statements following.
 	
bool mystery(const int **f, const int** e, int v)
{
if (f > e) return false;
auto p = (e - f) / 2 + f;
if (*p == v) return true;
if (v < *p) return mystery(f, p - 1, v);
return mystery(p + 1, e, v);
}
 	
if (f > e) is a ...
 	
if (*p ==v) is a ...
 	
if (v < *p is a ...
 	
mystery() implements the _ algorithm
 	
when called, mystery has infinite recursion (T/F)
 	
the mystery function is recursive (T/F)
 	
the mystery function has a stack overflow for certain inputs (T/F)
 	
the mystery function is very efficient (T/F)
	base case
	
	base case
	
	recursive case
	
	binary search
	
	F
	
	T
	
	F
	
	T
	
	NOTES: mystery fcn implements a binary search of an ordered array or vector. it does so by using pointers => very efficient. works correctly for any valid input

Which of the following is true about using recursion?	A recursive computation solves a problem by calling itself with simpler input

What is the value of r(3)?
 	
int r(int n)
{
if (n < 2) { return 1; }
return n * r(n - 1);
}
	6

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 pi. The formula states that pi squared over 6 is equal to the limit, as n goes to infinity, of the series 1/1 + 1/2^2 + 1/3^2 +...1/n^2.. 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);
	}

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; }

What is the value of r("ihihihihih")?
 	
int r(const string& s)
{
if (s.size() < 2) return 0;
return (s.substr(0, 2) == "hi") + r(s.substr(1));
}
	4

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

What is the value of r(3, 3)?
 	
int r(int n, int m)
{
if (m) return n * r(n, m - 1);
return 1;
}
	27

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

What is the value of r(6)?
 	
int r(int n)
{
if (n > 0) return n + r(n - 1);
return n;
}
	21

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);
}
	9

What does this function do?
 	
int mystery(int n)
{
if (n < 2) return 1;
return mystery(n-1) + mystery(n-2);
}
	computes the Fibonacci number n

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**el*lo"

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);
}
	2

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);
}
	3

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 pi. The formula states that pi squared over 6 is equal to the limit, as n goes to infinity, of the series 1/1 + 1/2^2 + 1/3^2 +...+1/n^2. Which statement below is the recursive case for a recursive implementation that approximates this infinite series?	return 1.0 / (number * number) + computePI(number - 1);

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**ell**o"

What is the value of r(126)?
 	
int r(int n)
{
if (n >= 10) return n % 10 + r(n / 10);
return n;
}
	9

What does this function do?
 	
int mystery(int n)
{
if (n == 1) return 1;
return n * mystery(n-1);
}
	computes the Factorial number n

Midterm 3

Study

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 inherits name and salary, but Manager functions can only change the values of the name data member

Examine the following UML diagram.
 	
Person Student
-name: string -studentID: long
+setName(string): void <== +Student(string, long)
+getName(): string +getID(): long
 	
Assume that the following code appears inside a member function or constructor of the Student class. Which of these statements would be legal??
name = "Bill Gates"; // I
setName("Bill Gates"); // II
name = name.substr(1); // III
studentID = 123L; // IV
studentID = getID() * 2; // V
	II, IV, and V

Inheritance gives your programs the ability to express _____ between classes.	f4(cout);
_____ 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
A classification hierarchy represents an organization based on _____ and _____.	generalization and specialization

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()

The AeroCar class is derived from Car and it overrides the setSpeed(double) member function. In the AeroCar setSpeed function, how can the Car setSpeed function be called?	Car::setSpeed(newSpeed)
Examine the following class hierachy: The ostream class is the/a ___ class of ofstream	base

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();
}	

Suppose that we have a Question class that contains two data members - a query and answer both of which are type string. The NumericQuestion class is derived from Question. Which of the following is true?	NumericQuestions contains both a query and an answer string
Which one of the following is an example of the "substitution principle"?	A derived class object can be used in place of a base-class object
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?	Question::display()
Examine the following UML diagram. Person Student -name: string -studentID: long +setName(string): void <== +Student(string, long) +getName(): string +getID(): long Which of these data members or member functions are inherited (and accessible) by the Student class??	getName(), setName()

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 Employee::getSalary();

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

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: 350

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;
};	

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
What is true about this inheritance hierarchy (using both C++ and traditional, classic OO terminology)? Assume that Party has a member function (which Person overrides), declared as: virtual void print() = 0;	Person is a derived class Organization is a superclass Organization is a subclass Party is an abstract class Party is a base class

Above is is a class hierarchy for different card games.
 	
class Hand {
std::vector<Card> cards;
public:
Hand() = default;
virtual ~Hand() = default;
void add(const Card&);
Card get(size_t index) const;
virtual int score() const = 0;
virtual void sort();
};
class PokerHand : public Hand { .. }
class BlackjackHand : public Hand { .. }
 	
Assuming that these are the only classes and that the concrete classes are correctly completed, which of the following non-member functions are polymorphic? More than one answer may be correct.
	void draw(Hand& h) const { ... }
	void draw(Hand* h) const { ... }
	
	NOTES: must use a pointer or a reference to the base class for polymorphic behavior

Below you'll find a C++ class hierarchy. All classes (including Card) are correctly and fully implemented.
 	
class Hand {
std::vector<Card> cards;
public:
Hand() = default;
virtual ~Hand() = default;
void add(const Card&);
Card get(size_t index) const;
virtual int score() const = 0;
virtual void sort();
};
class PokerHand : public Hand { .. }
class BlackjackHand : public Hand { .. }
 	
The Hand class:
 	
as a(n):
 	
relationship with vector:
 	
implements the accessor
 	
contains an abstract function
 	
is a derived class of
	NOTES: Hand class is an abstract class bc it has one pure-virtual (abstract) fcn score. It is a base class, not a derived class. It has a has-a relationship with vector, and implements the accessor function, get.
	
	abstract class
	
	has-a
	
	get
	
	score
	
	none

Putting the keyword final at the end of the class heading prohibits the creation of subsequent derived classes.	T
Composition can be used to create adapter classes that change the interface of one class to meet the needs of another.	T
Abstract classes provide a set of capabilities that derived classes my inherit.	F
In C++, private inheritance can be used to create adapter classes.	T
An abstract class may, but is not required to, override its pure virtual (abstract) member functions	F
Since an abstract class cannot be instantiated, it is illegal to have references of abstract types.	F

The public inheritance relationship is informally known as has-a.	F
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; Ducky duck;	pet = kit;

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(const Hand* h);

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

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:
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

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	Puppy* p = &pet;

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;
};
	dance()

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 { . . . };	

Examine the class hierarchy below. Assume that both derived classes are concrete and completely defined. Which line of code is illegal:
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

Here is a class hierarchy for different card games.
class Hand {
std::vector<Card> cards;
public:
Hand() = default;
virtual ~Hand() = default;
void add(const Card&);
Card get(size_t index) const;
virtual int score() const = 0;
virtual void sort();
};
class PokerHand : public Hand { .. }
class BlackjackHand : public Hand { .. }
 	
According to this design, which methods may the PokerHand and BlackjackHand use without making any changes? There may be more than one correct answer.
	add()
	get()
	sort()

If a class is abstract, you may create a pointer of that class.	T
Virtual member functions are implemented by adding a new pointer to every object that contains at least one virtual function.	T
In private inheritance a using declaration is employed to selectively bring base class members into the derived class scope.	T
The private inheritance relationship is informally known as implemented-with	T
Tell the compiler that you intend to override a base class function by adding the keyword override as an annotation before the function header	F
The public inheritance relationship is informally known as is-a.	T

What does this code mean?
class X : public Y {
. . .
};
	Every X object is-a Y object

What does this code mean?
class X : Y {
. . .
};
	Each X object is-implemented in terms of Y

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) { . . . }
	Code compiles, but the parameter is treated as a Hand object, not a PokerHand, so it is not drawn correctly.

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() {
Shape a = Shape(), b = Square(), c = Oval();
iam(a);
iam(b);
iam(c);
}
	ShapeShapeShape

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()

A(n) ___ is an occurrence of an undesirable situation that can be detected during program execution.	exception
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
The statements that may generate an exception are placed in a ___ block.	try
What happens when this code fragment runs?cout << stoi("12") << endl;	stoi)_ returns 12
The heading of a try block can contain ellipses in place of a parameter.	F

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 try block is followed by one or more ___ blocks.	catch
What happens when this code fragment runs? istringstream in(".5"); int n; in >> n;	It sets an error state in in

Which line fails to work correctly?
template <typename T>
void print(const T& item)
{
cout << item << endl;
}
	None of these

What header file to you need to include to use the standard C++ error-handling classes?	<stdexcept>
The C++11 standard library provides the function stoi() to convert a string to an integer. Which library is it found in?	cnvt
What happens when this code fragment runs in C++ 11? istringstream in("one"); int n; in >> n;	It sets an error state in in
What happens when this code fragment runs in C++ 11? cout << stoi("one") << endl;	It sets an error state in cout

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"; }	

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);

The line: ifstream in("x"); throws a runtime exception if a file x cannot be found.	F
Functions with generic parameters may use the keyword class or the keyword struct for their type parameters.	F
Calling a function like to_string(3.5) is known as implicit instantiation.	T
A template function may be declared in a header file but must be defined in an implementation file.	F
You can report a syntax error encountered in your code by using the throw keyword.	F
To use different versions of a function depending on the platform is called conditional compilation.	T
The #if preprocessor directive can compare integers.	T
The standard library version of sqrt(-2) throws a runtime exception because there is no possible answer.	F
The line: cin >> n; throws a runtime exception if n is an int and it tries to read the input "one".	F
Without try and catch, the throw statement terminates the running program.	T