C++ Midterm #3

Share

10 studiers today Leave the first rating

Terms in this set (473)

The following definition: vector<double> data;	creates a vector of size 0

[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;

[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

[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) . . .

[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)

[1301] Which line below points ppi to pi?
 	
int main()
{
double pi = 3.14159;
double *ppi;
// code goes here
// code goes here
}
	ppi = π

The following definition: vector<double> v{3, 5};	creates a vector of [3.0, 5.0]

[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 << &pi;
cout << ppi;
cout << &ppi;
cout << *pi;
```

None of these

[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;

[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.

[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;

[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)

[1503] 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->" << 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

Does not compile; e is undefined.

[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

[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;

[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;

[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;

The following definition: vector<double> v(3, 5);	creates a vector of [5.0, 5.0, 5.0]

[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

[1604] Below is a declaration for a partially-filled array. What is the correct prototype for a function back() that returns a reference to the last element?
 	
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);
	double& back(double a[], size_t size);

[1704] Where is the pointer s2 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

[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

[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;

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

[1305] Assume that ppi correctly points to pi. Which line prints the size (in bytes) of pi?
 	
int main()
{
double pi = 3.14159;
double *ppi;
// code goes here
// code goes here
}
	cout << sizeof(*ppi);

[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

[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

[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);

[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

[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);

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

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

<cstring>

[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

[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

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

[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

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

[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

[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;

[1507] 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) + 1, end(a)) << endl;
}
```

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

6

[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);

[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.

[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.

[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.

[1708] What happens here?

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

Crashes when run
Undefined behavior

"CS50"
"CS500"
"CS150CS50"

"CS50"

[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];

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

[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

[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)

[1609] 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 = 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.

Deletes elements from a partially-filled array

[1709] What happens here?

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

Crashes when run
Undefined behavior

"CS50"
"CS500"
"CS150CS50"

Undefined behavior

[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

[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

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

[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;

[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.

[1510] 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 + 1) << endl;
}
```

Does not compile
3
2
5
4

2

[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

[1710] What happens here?

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

Crashes when run
Undefined behavior

"CS50"
"CS500"
"CS150CS50"

"CS150CS50"

[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

[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"

[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++;

[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

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

[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';

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

[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;

[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';

[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

[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

[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

[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

[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

[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"

[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

What is stored in data after this runs? vector<int> data{1, 2, 3}; data.pop_back();	None of these
[1313] These pointer should point to "nothing". Which is not correctly initialized?	vector<int> *vp;

[1514] What is the correct prototype for mystery? (It is not supposed to modify the array.)
 	
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);
	int mystery(const int *a, size_t n);

[1614] Below is remove(), a template function that works with a partially-filled array. The function removes all copies of the argument e, returning the number of copies 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

Not all copies of e are necessarily removed

[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

[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)

[1314] These pointer should point to "nothing". Which is not correctly initialized?

```
Star *ps = NULL;

vector<int> *vp(0);

int *pi = nullptr;

double *pd{};
```

All are correctly initialized to point to nothing

All are correctly initialized to point to nothing

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]

[1315] Which of these is the preferred way to initialize a pointer so that it points to "nothing"?

int *pi = nullptr;

[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)

[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);

[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

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

---

[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

[1716] What prints here?

```
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

dog != dog

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

[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]

[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

What is stored in data after this runs? vector<int> data{1, 2, 3}; data.front();	[1, 2, 3]
[1317] All of these are legal C++ statements; which of them uses the C++ address operator? int a = 3, b = 4;	int *p = &a;
What is stored in data after this runs? vector<int> data{1, 2, 3}; data.back();	[1, 2, 3]
[1318] All of these are legal C++ statements; which of them uses the C++ reference declarator? int a = 3, b = 4;	int &x = a;
[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

[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

[1617] 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;

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

Every time the function is called, an array element is "lost"

[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

[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

[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];

[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.

[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]
What is stored in data after this runs? vector<int> data{1, 2, 3}; data.clear();	[]
[1319] All of these are legal C++ statements; which of them uses the C++ pointer declarator? int a = 3, b = 4;	int *p = &b;

[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

[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];

[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();

[1419] What is the equivalent array notation? int dates[10]; cout << *(dates + 2) << endl; dates[2] + 2 dates[2] + 2 dates[0] + 4 dates[2] &dates[2] dates[0] + 2	dates[2]
[1320] All of these are legal C++ statements; which of them uses the C++ dereferencing operator? int a = 3, b = 4;	int x = *p;
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
[1321] All of these are legal C++ statements; which of them uses indirection? int a = 3, b = 4;	int x = *p;

[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

[1520] What is printed here? (Assume all includes have been added.)

```
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

900

[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;

[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];

[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)

[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

[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

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

[1322] In C++, global variables are stored:

in the static storage area

[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

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

[1323] What is true about an uninitialized pointer?

Dereferencing it is undefined behavior

[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

[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

[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*

[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

[1623] Below is removeDupes(), a template function leaves only unique values in a partially-filled array. The function returns the number of elements removed. The 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

It removes some duplicates, but not all of them

[1723] Is p properly NUL-terminated when this function is called?

```
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

Yes, the terminator is copied as the condition fails

[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

[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

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

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

[1325] What is true about this code? int * choice;	choice contains an undefined address

[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

[1524] 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 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
	Returns the largest number in the array

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

[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

When inserting a value into a partially-filled array, in ascending order, the insertion position is the index of the first value larger than the value.	True
[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()

Which of these are true?
 	
int main()
{
vector<int> v{1, 2, 3};
for (auto i = v.size() - 1; i >= 0; i--)
cout << v[i] << " ";
cout << endl;
}
	Endless loop (will likely crash, but not necessarily)
	
	Issues a compiler warning, but no error
	
	Prints 3 2 1

[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

[1326] How can we print the address where n is located in memory? int n{500};	cout << &n << endl;

[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

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

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

[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

[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

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

True

[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()

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.

True

[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()

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;

[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

[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;

[1328] What is a common pointer error?	Using a pointer without first initializing it

[1528] 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;
}
None of these
4
3
1
2
	4

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

[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()

[1329] What is printed when you run this code? int x(100); cout << &x << endl;	The memory location where x is stored
[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
C-strings use the strcpy() function for concatenation.	False

Which defines a vector to store the salaries of ten employees?	vector<double> salaries(10);
When deleting elements from a partially-filled array, the array should not be declared const.	True
[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)

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

[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

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

In a partially-filled array size represents the number of elements that are in use.	True
The strncat() function is tricky to use correctly.	True

[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

[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])

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

[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

When inserting a value into a partially-filled array, elements following the insertion position are shifted to the right.	True
You can compare two C-strings, s1 and s2, by using the strcmp() function.	True
[1332] What is printed when you run this code? int *n{nullptr}; cout << n << endl;	The address value 0

[1532] 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;
}
4
5
2
None of these
0
	5

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

a3
a1
None of these
a5
a2

a2

[1333] What is printed when you run this code?

```
int *n{nullptr};
cout << *n << endl;
```

No compilation errors, but undefined behavior

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

C-strings are character arrays that rely on a special embedded sentinel value, the character with the ASCII code 0.

True

[1334] What is printed when you run this code?

```
int *n{nullptr};
cout << &n << endl;
```

The address value where n is stored

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

The allocated size for the C-string char s1[] = "hello"; is 6 characters, while the effective size is 5 characters.

True

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

True

[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

[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

[1335] What is printed when you run this code? int *p = &0; cout << *p << endl;	No output; compiler error.
The sizeof operator returns the allocated size of a C-string allocated as an array.	True
When inserting an element into a partially-filled array, it is an error if size >= capacity.	True

[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.
	Returns the smallest number in the array

[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
[1336] What is printed when you run this code? int n{}; int *p; *p = &n; cout << *p << endl;	Will not compile

[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.

In a partially-filled array, the size may be less than the array's capacity.	True
The effective size of the C-string char * s1 = "hello"; is 5 characters, but 6 characters are used for storage.	True

Which line compiles, but crashes when run?
 	
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.
}
	4

[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
Which statement is false? The elements in a vector: Are accessed by using an index or subscript Each use the same amount of memory Are are all of the same type Are homogeneous	None of these
strcmp(s1, s2) returns a positive number if s1 is lexicographically "greater than" s2.	True
When comparing two partially-filled arrays for equality, both arrays should be declared const.	True

[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
	Undefined. Depends on the input.

[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

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

[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
In C++ the parameterized collection classes are called _____?	templates
[1338] What is the term used to describe a variable with stores a memory address?	pointer
An incomplete type and a forward reference generally mean the same thing.	True

[1537] What is the name for this algorithm?
 	
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
	A fencepost algorithm

When deleting an element from a partially-filled array, it is an error if the index of the element to be removed is >= size.	True
C-strings use the strcat() function for concatenation.	True
The strlen() function returns the effective size of a C-string.	True

[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]

Classes that contain objects as elements are called?	collections
[1339] Which of these is not one of the three characteristics of every variable?	alias
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
In a partially-filled array, the size represents the effective size of the array.	True
[1340] Which area of memory is your program code stored in?	Text

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()
	cout << speed.at(speed.size());

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

[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.

When inserting elements into a partially-filled array, the array should not be declared const.	True
C-strings are often needed to interoperate with legacy C libraries.	True
When inserting a value into a partially-filled array, in ascending order, the insertion position may be the same as capacity.	False
[1341] Which area of memory are local variables stored in?	Stack
vector<int> v;	Creates the empty vector []
You must use an integral constant or literal to specify the size of a built-in C++ array.	True
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.	True

[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]

The characters for the C-string char s1[] = "hello"; are stored in user memory and may be modified.	True
When inserting elements into a partially-filled array, the array should be declared const.	False

[1541] 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, 0)) << endl;
[0]
Does not compile. Arrays cannot be 0 length.
[]
[1]
No output
	[]

The reinterpret_cast instruction changes way that a pointer's indirect value is interpreted.	True
vector<int> v(1);	Creates the vector [0]
[1342] Which area of memory are global variables stored in?	Static storage area
v.begin()	Points to the first element in v
C-strings are char pointers to the first character in a sequence of characters, terminated with a '\0' character.	True
When comparing two partially-filled arrays for equality, both arrays should not be declared const.	False
Elements always allocated on the heap	vector
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

[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

C-string functions may be more efficient than C++ string member functions.	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
C++ arrays have no support for bound-checking.	True

[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

v.back();	Returns a reference to the last element in v
An array passed to a function f(int *a, ...) may have its elements changed.	True
v.erase(v.begin());	Removes the first element in v and shifts the rest to the left
strcmp(s1, s2) returns a negative number if s1 is lexicographically "less than" s2.	True
When inserting a value into a partially-filled array, elements following the insertion position are shifted to the left.	False
The parameter declarations int p* and int[] p mean the same thing.	False
In C++ assigning one array to another is illegal	True

[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

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

[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);

v.pop_back()	Removes the last element in v
Given the C-string char * s3 = "hello"; strlen(s3) returns 5.	True
How arrays are passed to functions	by address
In a partially-filled array, the size represents the allocated size of the array.	False
C-string assignment uses the strcpy() function.	True
In a partially-filled array, the capacity represents the effective size of the array.	False
What happens to an array when passed to a function	decays
The size of the array is not stored along with its elements.	True

[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);

v[3];	Returns a reference to the fourth element in v with no range checking
In a partially-filled array, all of the elements are not required to contain meaningful values	False
strcmp(s1, s2) returns 0 if s1 and s2 contain the same characters.	True
[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;
vector<int>v(2,3);	Creates the vector [3,3]
const int *array	Elements may not be modified; pointer may be
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
The C-string type is built into the C++ language, not defined in the standard library.	True
When inserting an element into a partially-filled array, it is an error if size < capacity.	False
int * const array	Elements in may be modified; pointer may not
The subscripts of a C++ array range from 0 to the array size - 1.	True
[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;

vector<int>v[2, 3];	Creates the vector [2, 3]
const int * const array	Neither pointer nor elements in may be modified
The strcpy() function always appends a trailing NUL when the copy is finished.	True
C++ arrays have no built-in functions for inserting and deleting.	True
[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;
v.push_back(3);	Adds a new element to the end of v
In a partially-filled array, all of the elements contain meaningful values	False
When deleting elements from a partially-filled array, the array should be declared const.	False
v.at(3);	Safely returns a reference to the fourth element in v
[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++;
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
sizeof(a) / sizeof(a[0])	Elements in array using arithmetic
The strncpy() function can be used to make sure that you don't copy more characters than necessary.	True
Programs written for embedded devices often use C-strings rather than the C++ library string type.	True
end(a) - begin(a)	Elements in array using pointer difference
You can create vector objects to store any type of data, but each element in the vector must be the same type.	True
Used to access the data inside a variable	variable name
The elements of a C++ array created in a function are allocated on the stack.	True
In a partially-filled array capacity represents the number of elements that are in use.	False
Assume the vector v contains [1, 2, 3]. v.erase(0); changes v to [2, 3].	False
The length of a C-string is never stored explicitly	True
When searching for the index of a particular value in a partially-filled array, the array should not be declared const.	False
for (auto e : a) . .	A range-based loop
The elements of a C++ array created outside of a function are allocated in the static-storage area.	True
Determines the amount of memory required and the operations permitted on a variable	variable type
The C-string literal "cat" contains 4 characters.	True
x = 0; for (auto e : a) x += e;	Cumulative algorithm
The elements of a C++ string array with no explicit initialization, created in a function will be set to the empty string.	True
The meaning assigned to a set of bits stored at a memory location	variable value

Assume vector<int> v; Writing cout << v.front(); throws a runtime exception.	True
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
Assume the vector v contains [1, 2, 3]. v.erase(v.begin() + 2); changes v to [1, 2].	True
An object whose value is an address in memory	pointer
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
x = a[0]; for (auto e: a) if (e > x) x = e;	Extreme values algorithm
The strncat() function allows you to limit the maximum number of characters that are concatenated.	True
The declaration: vector<string> v(5, "bob"); creates a vector containing five string objects, each containing "bob".	True
Expression using the address operator	p = &a;
In C++ printing an array name prints the address of the first element in the array.	True
auto p = a; while (p != end(a)) p++;	Iterator-based loop
The character with the ASCII code 0 is called the NUL character	True
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.	True
In the declaration: vector<int> v; the word int represents the object's base type.	True
Expression using the reference declarator	int x = 3;
cout << a[0]; while (i < len) cout << ", " << a[i++];	Fence-post algorithm
The characters for the C-string char * s1 = "hello"; are stored in user memory and may be modified.	False
strcmp(s1, s2) returns true if s1 and s2 contain the same characters.	False
An array passed to a function decays to a pointer.	True
In C++ initializing an array with the contents of another is illegal.	True
Expression using the dereferencing operator	y = *a;
The elements of a vector are allocated contiguously.	True
An array passed to a function f(int * const a, ...) may have its elements changed.	True
The strlen() function returns the allocated size of a C-string allocated as an array.	False
C++ arrays produce undefined results if you access an element outside the array.	True
vector subscripts begin at 0 and go up to the vector size - 1	True
Expression using the pointer declarator	double * v;
The C-string type is part of the standard library, not built into the C++ language.	False
The elements of an array may be allocated on the stack.	True
Explicitly initializing an array like this: int a[] = {1, 2, 3}; works in all versions of C++.	True
Expression returning the number of allocated bytes used by an object	sizeof(Star)
The clear() member function removes all the elements from a vector.	True
If p points to the first element in [1, 3, 5] then cout << ++*p prints 2.	True
You may use any kind of integral variable to specify the size of a built-in C++ array.	False
Address value 0	nullptr
The statement v.insert(v.end() + 1, 3) is undefined because end() + 1 points past the last element in the vector.	True
C-string assignment uses the = operator.	False

The elements of a C++ string array with no explicit initialization, created in a function will be set to null.	False
The length of a C-string is stored explicitly in its length data member	False
The library function begin(a) returns a pointer to the first element in the array a.	True
The statement v.insert(v.end(), 3) appends the element 3 to the end of the vector v.	True
The elements of an array may be allocated in the static storage area.	True
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.	False
Contiguous allocation means that the elements are stored next to each other in memory.	True
The allocated size for the C-string char s1[1024] = "hello"; is 6 characters, while the effective size is 5 characters.	False
C-string assignment uses the strcat() function.	False
Arrays generally have higher performance than a vector.	True
In C++ using == to compare one array to another is illegal.	False
The push_back member function adds elements to the end of a vector.	True
Assume the vector v contains [1, 2, 3]. v.erase(v.begin()); changes v to [2, 3].	True
The allocated size of a built-in C++ array may be changed during runtime	False
The function mystery(const int**, const int**) likely employs an iterator loop.	True
The strcat() function cannot overflow the storage allocated for the destination buffer.	False
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);	False
The strncpy() function always appends a trailing NUL when the copy is finished.	False
The expression begin(a) + 1 returns a pointer to the second element in the array a.	True
The declaration: vector<int> v(10); creates a vector object containing ten elements initialized to 0.	True
strcmp(s1, s2) returns a negative number if s1 is lexicographically "greater than" s2.	False
Array subscripts are not range checked	True
The reinterpret_cast instruction produces a temporary value by converting its argument.	False
Assume the vector v contains [1, 2, 3]. v.pop_back(); changes v to [1, 2].	True
The sizeof operator returns the effective size of a C-string allocated as an array.	False
An array passed to a function is passed by address.	True
The term for classes with a base-type specification are parameterized classes.	True
In C++ initializing an array with the contents of another is permitted.	False
Assume that v contains [1, 2, 3]. The result of writing cout << v[4]; is undefined.	True
The strncpy() function is straightforward and easy to use.	False
If size_t len = 0; then len - 1 is the largest possible unsigned number.	True
C++ arrays use bound-checking when you access their elements with the at() member function.	False
strcmp(s1, s2) returns a positive number if s1 is lexicographically "less than" s2.	False
If p points to the first element in [1, 3, 5] then cout << *++p prints 3.	True
The elements of a C++ array created in a function are allocated on the heap.	False
The C++ term for classes like vector are template classes.	True

The algorithm that finds the address of the smallest element in an array is called an extreme values algorithm.	True
You can compare two C-strings, s1 and s2, by using the == operator.	False
In C++ assigning one array to another is permitted.	False
A vector subscript represents the element's offset from the beginning of the vector.	True
C-strings use the + operator for concatenation.	False
The expression **p++ means the same as** (p++).	True
C++ arrays throw an out_of_bounds exception if you access an element outside the array.	False
The declaration: vector<string> v{"bill", "bob", "sally"}; creates a vector containing three string objects.	True
C-strings are char pointers to the first character in a sequence of characters, terminated with a '0' character.	False
Before passing an array to a function, sizeof(a)/sizeof(a[0]) will tell the number of elements in the array.	True
In C++ an array variable and the array elements are separate. The array variable contains the address of the first element in the array.	False
The declaration: vector<int> v(10, 5); creates a vector object containing ten integers.	True
For systems programming (such as operating systems), arrays are used more often than vectors.	True
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.	False
In C++ printing an array name prints the value of the first element in the array.	False
Assuming that Star is a structure, the declaration: vector<Star> stars(3); creates three default initialized Star objects.	True
The elements of a C++ int array with no explicit initialization, created in a function will be set to zero.	False
The library function end(a) returns a pointer to position right past the last element in the array a.	True
The C-string literal "cat" contains 3 characters.	False
The declaration: vector<string> v(5); creates a vector containing five empty string objects.	True
C++ arrays can be allocated with a size of 0.	False
Assume the vector v contains [1, 2, 3]. v.erase(0); is a syntax error.	True
For embedded systems, arrays are preferred over vector.	True
The strcpy() function expands the destination string to make sure it is large enough to hold the source string.	False
The declaration: vector<int> v; creates a vector object with no elements.	True
The static_cast instruction changes way that a pointer's indirect value is interpreted.	False
The parameter declarations int *p and int p[] mean the same thing.	True
A vector represents a linear homogeneous collection of data.	True
The size of the array is stored along with its elements.	False
The algorithm that prints elements separated by commas is called the fencepost algorithm.	True
Assume vector<double> v; Writing cout << v.back(); is undefined.	True
The allocated size of a built-in C++ array may be changed during runtime	False
The elements of a vector are allocated on the heap.	True
A vector variable may be allocated on the stack.	True

A forward reference can be used when you want to use a structure as a data member without first defining the entire structure.	False
Elements in a vector are accessed using a subscript.	True
Before passing an array to a function, sizeof(a) will tell you the array's allocated size, but not the number of elements.	True
The elements of a C++ array created outside of a function are allocated on the stack.	False
Assume that v contains [1, 2, 3]. The result of writing cout << v.at(4); throws a runtime exception.	True
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
The statement v.insert(v.begin(), 3) inserts the element 3 into the vector v, shifting the existing elements to the right	True
After passing an array to a function, sizeof(a)/sizeof(a[0]) will tell the number of elements in the array.	False
Vector subscripts begin at 1 and go up to the vector size.	False
If p points to the first element in [1, 3, 5] then cout << *++p prints 1.	False
C++ arrays offer built-in member functions for inserting and deleting.	False
If p points to the first element in [1, 3, 5] then cout << ++*p prints 1.	False
Explicitly initializing an array like this: int a[] = {1, 2, 3}; only works in C++ 11.	False
The statement v.insert(v.end(), 3) is undefined because end() points past the last element in the vector.	False
The library function begin(a) returns a pointer to the element right before the first in the array a.	False
Assume that v contains [1, 2, 3]. The result of writing cout << v.at(4); is undefined.	False
For embedded systems, vector is preferred over arrays.	False
The C++ term for classes like vector are generic classes.	False
The statement v.insert(v.begin(), 3) inserts the element 3 into the vector v, overwriting the exiting element at index 0.	False
For systems programming (such as operating systems), vectors are used more often than arrays.	False
The push_back member function adds elements to the end of a vector as long as there is room for the elements.	False
For an equivalent number of elements, a vector will use less memory than an array.	False
The declaration: vector<int> v(10); creates a vector object containing uninitialized elements.	False
The expression **p++ means the same as (**p)++.	False
The declaration: vector<int> v(10, 5); creates a vector object containing five integers.	False
An array passed to a function f(const int *a, ...) may have its elements changed.	False
The elements of a vector may be allocated on the stack.	False
The declaration: vector<string> v(5); creates a vector containing five null pointers.	False
For an equivalent number of elements, a vector will use more memory than an array.	False
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
The algorithm that prints elements separated by commas is called a cumulative algorithm.	False
Assume that v contains [1, 2, 3]. The result of writing cout << v.at(4); is a compiler error.	False
The algorithm that finds the position of the largest element in an array is called a cumulative algorithm.	False

The algorithm that finds the position of the largest element in an array is called a cumulative algorithm.	False
Vector subscripts begin at 1 and go up to the vector size.	False
After passing an array to a function, sizeof(a) will tell you the array's allocated size, but not the number of elements.	False
A vector consists of named members.	False
If p points to the first element in [1, 3, 5] then cout << *p++ prints 3.	False
The declaration: vector<int> v(10, 5); is illegal.	False
The library function end(a) returns a pointer to the last element in the array a.	False
Assume vector<double> v; Writing cout << v.back(); throws a runtime exception.	False
A vector generally has higher performance than an array.	False
Assume that v contains [1, 2, 3]. The result of writing cout << v[4]; is a compiler error.	False
If size_t len = 0; then len - 1 is the smallest possible unsigned number.	False
The declaration: vector<int> v = new vector<>(); creates a vector object with no elements.	False
The expression begin(a) + 1 returns a pointer to the first element in the array a.	False
The pop_back member function adds elements to the end of a vector.	False
The function mystery(const int**, const int**) likely employs a counter-controlled loop.	False
An array passed to a function is passed by reference.	False