

Note: Some comments in code may be incorrect. This is to simulate an employee who thinks the code does X, but it does Y.

Note: Unless specified, there may be more than one correct answer

**You will have two hours to complete this exam.**

1. Suppose we implement a queue.
  - a. What is the worst-case time to dequeue?
    - i.  $O(n^2)$
    - ii.  $O(n)$
    - iii.  $O(1)$
    - iv.  $O(2^n)$
  - b. What is the worst-case time to enqueue?
    - i.  $O(n^2)$
    - ii.  $O(n)$
    - iii.  $O(1)$
    - iv.  $O(2^n)$
  - c. What is the worst-case time to peek?
    - i.  $O(n^2)$
    - ii.  $O(n)$
    - iii.  $O(1)$
    - iv.  $O(2^n)$
  - d. What is the worst-case time of isEmpty?
    - i.  $O(n^2)$
    - ii.  $O(n)$
    - iii.  $O(1)$
    - iv.  $O(2^n)$
2. Suppose we implement a Stack.
  - a. What is the worst-case time to dequeue?
    - i.  $O(n^2)$
    - ii.  $O(n)$
    - iii.  $O(1)$
    - iv.  $O(2^n)$
  - b. What is the worst-case time to enqueue?
    - i.  $O(n^2)$
    - ii.  $O(n)$
    - iii.  $O(1)$
    - iv.  $O(2^n)$
  - c. What is the worst-case time to peek?
    - i.  $O(n^2)$

- ii.  $O(n)$
  - iii.  $O(1)$
  - iv.  $O(2^n)$
- d. What is the worst-case time of isEmpty?
  - i.  $O(n^2)$
  - ii.  $O(n)$
  - iii.  $O(1)$
  - iv.  $O(2^n)$
- 3. Suppose we have a word W and the length L. We want to see if W is a palindrome (radar backwards is radar). Which of the following is the correct implementation?
  - a. A
  - b. B
  - c. C

A)

```
bool isPalindromeA(string word){
    queue<char> q;

    // Put all of the word in the queue, reversed
    for (char& c : word){
        q.push(c);
    }

    char checkC;

    // Check the reversed word against the orig
    word
    for (char& c : word){
        checkC = q.front();
        q.pop();

        if (c != checkC){
            return false;
        }
    }

    return true;
}
```

B)

```
bool isPalindromeB(string word){
    // Go through the word, in linear time
    for (size_t i = word.size() - 1, j = 0;
        i >= j;
        i--, j++){
        if (word[i] != word[j]){
            return false;
        }
    }

    return true;
}
```

C)

```
bool isPalindromeC(string word) {
    stack<char> s;
    for (int i = word.size()/2;
        i >= 0;
        i--) {
        s.push(word[i]);
    }

    for (auto i = word.size() - 1;
        i >= word.size()/2;
        i--) {
        if (s.top() != word[i]) {
            return false;
        }
        s.pop();
    }

    return true;
}
```

4. What is wrong with the following tree?

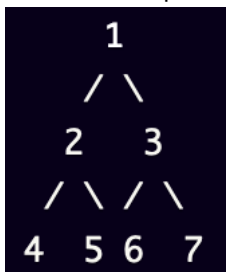
- Here is a binary search tree (BST) that can be constructed with the input numbers 12, 5, 3, 85, 39, and 90:



Credit: GPT-3.5 Turbo

- a. Nothing
- b. 85 and 90 should be swapped
- c. The root is incorrect
- d. Something else (describe below)

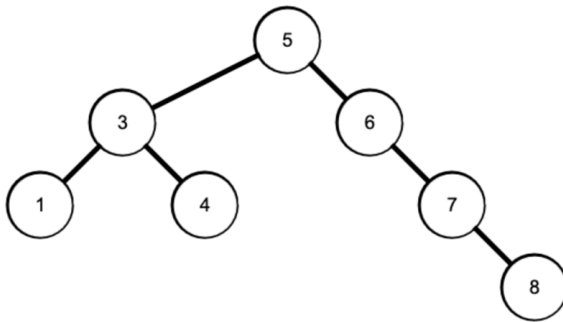
5. What input would produce the following tree?



Credit: CPT-3.5 Turbo

- a. I cannot know without more details
- b. 7 6 5 4 3 2 1
- c. 1 2 3 4 5 6 7
- d. 3 2 1 7 6 5 4
- e. Something else (describe below)

Use the tree below for 6 & 7.



Credit: <https://wintercore.github.io/bst-visualizer/>

6. What is the insert order to obtain this tree?
  - a. 8 7 4 1 3 6 5
  - b. 5 3 6 1 4 7 8
  - c. 5 6 3 7 4 1 8
  - d. 6 5 7 8 4 3 1
  - e. Something else (Describe to the right)
7. What insert order can we use to obtain a balanced tree?
  - a. No such order exists
  - b. The order:
8. Why would we want to write a private and public version of the same function?
  - a. We do not, it wastes time
  - b. If we are working with time-critical programs where 1ns is critical
  - c. To increase security
  - d. To hide pointer information
  - e. To convert a const parameter to non-const parameter
  - f. Because our boss tells us to
9. What should every class contain? [Select exactly 3]
  - a. Constructor
  - b. Setter
  - c. Getter
  - d. Destructor
  - e. Overloaded stream operators (<< and/or >>)
  - f. Overloaded assignment operator
  - g. Inheritance
  - h. Copy constructor
  - i. Private members
  - j. Virtual functions
  - k. Virtual constructors

10. Suppose we have a queue with 100 elements.

- a. When we dequeue,
  - i. Only the front changes
  - ii. Only the rear changes
  - iii. Nothing changes
  - iv. Other/Conditionally I, II, III
- b. When we enqueue,
  - i. Only the front changes
  - ii. Only the rear changes
  - iii. Nothing changes
  - iv. Other/Conditionally I, II, III
- c. When we peek at the front or back,
  - i. Only the front changes
  - ii. Only the rear changes
  - iii. Nothing changes
  - iv. Other/Conditionally I, II, III

11. What is wrong with the following code?

```
// The base class with x, y, and z visible
```

```
class BaseClass {
```

```
protected:
```

```
    int x; int y; int z;
```

```
};
```

```
// The mid class with x, y, z, a, b, and c visible
```

```
class MidClass: BaseClass {
```

```
protected:
```

```
    int a; int b; int c;
```

```
};
```

```
// The upper clas with x, y, z, a, b, c, d, f, and g visible
```

```
class UpperClass: MidClass {
```

```
    UpperClass() {
```

```
        x = d; y = f; z = g;
```

```
    }
```

```
protected:
```

```
    int d; int f; int g;
```

```
};
```

**Write what is wrong with the code here (also provide solution):**

12. Write the purpose of the following data structures:
- a. Tree
  - b. Queue
  - c. List
  - d. Stack
13. Virtual functions must be pure virtual functions if there is at least one pure virtual function.
- a. True
  - b. False
14. We use the @Override attribute to override (not overload) a function.
- a. True
  - b. False
15. Inserting at the end of a singly linked list is *a/ways* slower than inserting at the end of an array.
- a. True
  - b. False
16. We use UML to graphically describe the functionality of a system.
- a. True
  - b. False
17. UML can only describe logical, class, and other code-related relationships.
- a. True
  - b. False
18. You attempt to allocate memory, but it fails. An exception (`std::bad_alloc`) is thrown. But you are a good programmer and anticipated this being thrown, write the code below. **You want to print out the debug info using `.what()`.**

19. FIFO is the same as FILO

- a. True
- b. False

20. FIFO is comparable as FCFS

- a. True
- b. False

21. We can use a BST to sort, reverse, and maintain order of an input vector.

- a. We can only maintain the order
- b. We can only reverse it, the original order is lost
- c. We can only sort it, the original order is lost
- d. We can do all of them

22. FIFO is used in the following data structures:

- a. Tree
- b. Queue
- c. List
- d. Stack

23. FILO is used in the following data structures:

- a. Tree
- b. Queue
- c. List
- d. Stack

24. Find any issues with the enqueue code.

```
Queue enqueue(T value) {  
    Node* newNode = new Node(value)  
    if (rear == nullptr) {  
        front = rear = newNode;  
    } else {  
        rear->next = newNode;  
        rear = newNode;  
    }  
}
```

25. Find any issues with the dequeue code.

```
void dequeue(T out) {  
    if (front == nullptr) {  
        abort("front is null ptr");  
    }  
  
    out = front;  
    front = front->next;  
    delete temp;
```

```
        if (front == nullptr) {  
            rear = nullptr;  
        }  
    }
```

26. What is the proper way to cast an int to a double?

- a. `int x = y;`
- b. `int x = (int)y;`
- c. `int x = static_cast<int>(y);`
- d. `int x = dynamic_cast<int>(y);`

27. It is valid to use C code in C++

- a. True
- b. False

28. It is valid to use C++ code in C

- a. True
- b. False

29. Which line is correct to override a pure virtual function in C++?

- a. `void f() override {}`
- b. `override void f() {}`
- c. `void override f() {}`
- d. `void f() {} override`
- e. `@Override void f() {}`

30. What does delete do?

- a. Deallocate memory
- b. Allocate memory
- c. Call the constructor
- d. Call the destructor
- e. None of the above (describe correct answer)

31. What does new do?

- a. Deallocate memory
- b. Allocate memory
- c. Call the constructor
- d. Call the destructor
- e. None of the above (describe correct answer)

32. In C++, we can inherit from 500 base classes

- a. True
- b. False

33. In C++, what problems do we encounter with inheritance?

- a. None
- b. Tree Problem



- c. Circular Problem
- d. Diamond Problem
- e. Ruby Problem

34. What is wrong with this BST's private insert function?

Specifications:

- There must be a public interface which the developer will call insert(50). The caller should expect a Boolean value which indicates if it failed to insert. It does not matter if it failed to allocate memory or if it already exists.
- Within the private function, we want to set the left and right. It should return the same value as the public interface.
- The node contains getters and setters for the left and right node. The function to get the value is value().

/// Note: The public function is good

```
Node* insertRec(Node* root, int value) {
    if (root == nullptr) {
        return new Node(value);
    }

    if (key < root->value()) {
        root->setLeft = insertRec(root->left, key);
    } else if (key > root->key) {
        root->setRight = insertRec(root->right, key);
    }

    return root == nullptr;
}
```

**Describe here:**

35. What is the correct order for in-order transversal?

- a. LEFT DATA RIGHT
- b. LEFT RIGHT DATA
- c. RIGHT LEFT DATA
- d. RIGHT DATA LEFT
- e. DATA LEFT RIGHT
- f. DATA RIGHT LEFT

36. What is the correct order for pre-order transversal?

- a. LEFT DATA RIGHT
- b. LEFT RIGHT DATA
- c. RIGHT LEFT DATA

- d. RIGHT DATA LEFT
  - e. DATA LEFT RIGHT
  - f. DATA RIGHT LEFT
37. What is the correct order for post-order traversal?
- a. LEFT DATA RIGHT
  - b. LEFT RIGHT DATA
  - c. RIGHT LEFT DATA
  - d. RIGHT DATA LEFT
  - e. DATA LEFT RIGHT
  - f. DATA RIGHT LEFT
38. When do we use pre-order traversal?
39. When do we use post-order traversal?
40. When do we use in-order traversal?
41. We can use a queue to implement an array.
- a. True
  - b. False
42. `this` keyword is an object.
- a. True
  - b. False
43. When we overload an operator, we can provide the precedence.
- a. True
  - b. False
44. Within an overloaded operator, say for `+`, we must use the `+` symbol within the body, otherwise we get a compiler error.
- a. True
  - b. False
45. Suppose class A and B are defined. B inherits from A as such `class B : A {...};`. Now, we write `B newClass = A()`. This is valid.
- a. True
  - b. False

46. Suppose class A and B are defined. B inherits from A as such `class B : A {...};`. Now, we write `A newClass = A()`. This is valid.
- True
  - False
47. Suppose class A and B are defined. B inherits from A as such `class B : A {...};`. Now, we write `A newBClass = B()`. This is valid.
- True
  - False
48. Suppose class M is fully functional and has a destructor. Within M, we allocate memory that needs to be destroyed. We declare it as `M* m = new M()`. Now, we are done with m and destroy it like `free(m)`. The following occurs:
- Compiler Error
  - Runtime Error
  - Memory Leak
  - Dangling Pointer
  - No Error
49. Suppose class M is fully functional and has a destructor. Within M, we allocate memory that needs to be destroyed. We declare it as `M* m = new M()`. Now, we are done with m and destroy it like `~(*m); free(m)`. The following occurs:
- Compiler Error
  - Runtime Error
  - Memory Leak
  - Dangling Pointer
  - No Error
50. Suppose class M is fully functional and has a destructor. Within M, we allocate memory that needs to be destroyed. We declare it as `M* m = new M()`. Now, we are done with m and destroy it like `m->~M(); free(m)`. The following occurs:
- Compiler Error
  - Runtime Error
  - Memory Leak
  - Dangling Pointer
  - No Error
51. All data structures are linear because non-linear structures are too complicated to program with efficiency.
- True
  - False
52. Polymorphism is the same as inheritance.
- True
  - False

53. What is required for a class to be considered abstract?

- a. It has at least one *virtual* function
- b. It has more than one *virtual* function
- c. It has all *pure virtual* functions
- d. It has at least one *pure virtual* function
- e. It has more than one *pure virtual* function
- f. It has *virtual* constructors
- g. It has *virtual* destructors

54. Is there anything wrong with these constructors? If so, what is wrong?

```
virtual MyClass() {  
    cout << "Setting up..." << endl;  
    ... set all members ...  
}  
virtual MyClass(MyClass* copy) {  
    cout << "Copying class..." << endl;  
    ... copy over members ...  
}
```

55. What will be implicitly called when the operator is invoked?

```
operator<<(ostream& lhs, Data& rhs);
```

56. What is wrong with the implementation below?

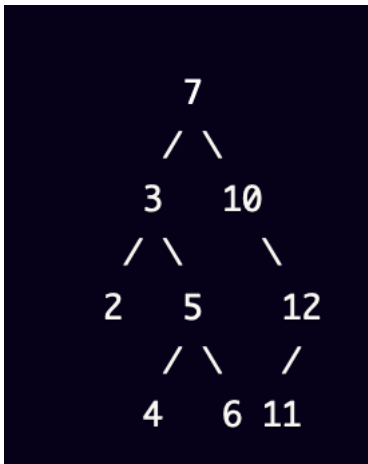
```
ostream operator<<(ostream& lhs, Data& rhs) {  
    lhs << rhs.data() << endl;  
}
```

57. What is wrong with the following reverse function (of a singly linked list)?

Note: The members of Node are public

```
Node* reverse(Node* cur) {  
    Node * last = cur;  
    if (cur != nullptr) {  
        last->next = nullptr;  
        cur = reverse(cur);  
        cur->next = last;  
    }  
    return cur;  
}
```

58. When would we want to use polymorphism?
- When we want to look fancy
  - When we want to use more generic base classes
  - When we want to use write less code to do more
59. When we inherit from multiple classes, how do we access a function in B1 instead of B2 from C `class C: B1, B2 {...};` (From the perspective of C)?
- `dynamic_cast<B1>(this)->f1();`
  - `((B1)(this))->f1();`
  - `__treat_as_base__((B1))  
this->f1();`
  - `this->B1.f1();`
  - `this->B1->f1();`
  - Something else
60. Who can access private members of class A?
- private, protected, public members of A
  - private members of B (which inherits from A)
  - private, protected, public members of B (which inherits from A)
  - friends of A
  - all classes which inherit via public or protected
61. Which of the following protection levels are valid in C++?
- private
  - protected
  - public
  - fileprivate
  - internal
62. What is the output of the code below?
- `preOrderTraversal()`
  - `postOrderTraversal()`
  - `inOrderTraversal()`



Credit: GPT-3.5 Turbo

63. What is the output of the following code?

```
/// Node has last, next, and data
void foo(Node * n) {
    for (; n != nullptr; n = n->next) {
        cout << n->data() << endl;
    }
}
```

64. What is happening in the following code?

```
/// Node has last, next, and data
void bar(Node * n) {
    Node * p = nullptr;
    for (; n != nullptr; n = n->next) {
        if (p) {
            barfoo(p, n); // Assuming we are never at list end
            p = nullptr;
        } else {
            p = n;
        }
    }
}

void barfoo(Node * p, Node * n) {
    Node * pL = p->last,
        * pN = p->next,
        * nL = n->last,
        * nN = n->next;

    if (pL) {
        pL->next = n;
    }
    if (pN) {
        pN->last = n;
    }
    if (nL) {
        nL->next = p;
    }
    if (nN) {
        nN->last = p;
    }
    n->last = p->last;
    n->next = p->next;
    p->last = nL;
    p->next = nN;
}
```