Name: ______________________________

2143 OOP Exam 2
Fall 2025

1. (1 point) What is a base class in C++?
   - A. A class that inherits from another class
   - **B. A class that is inherited from by other classes**
   - C. A class with only private members
   - D. A class that cannot be instantiated

2. (1 point) Which best describes a derived class?
   - A. A class that defines a virtual method
   - B. A class that cannot be inherited
   - **C. A class that inherits members from a base class**
   - D. A class that only contains abstract methods

3. (1 point) What is function overloading in C++?
   - **A. Defining multiple functions with the same name but different parameters**
   - B. Replacing a base class method in a subclass
   - C. Defining multiple classes with the same name
   - D. Adding parameters to a constructor

4. (1 point) What is function overriding?
   - A. Defining multiple constructors in a class
   - **B. Redefining a base class's virtual function in a derived class**
   - C. Creating multiple classes with shared base methods
   - D. Hiding a base class variable

5. (1 point) What does polymorphism allow in C++?
   - A. Only overloading operators
   - B. Creating multiple constructors with different names
   - **C. Using a single interface to represent different types**
   - D. Avoiding dynamic memory allocation

6. (1 point) What is the purpose of a virtual function?
   - **A. To allow derived classes to override the function for polymorphism**
   - B. To declare a function without implementation
   - C. To prevent a function from being overridden
   - D. To share functions between unrelated classes

7. (1 point) What is a pure virtual function?
   - A. A function defined with 'static'
   - **B. A virtual function with no implementation in the base class**
   - C. A function that cannot be overridden
   - D. A constructor in an abstract class

8. (1 point) Which of the following makes a class abstract?
   - A. Having at least one static method
   - B. Inheriting from more than one class
   - **C. Having at least one pure virtual function**
   - D. Having no constructors

9. (1 point) What is a concrete class?
   - **A. A class that can be instantiated and has complete implementations**
   - B. A class with only virtual functions
   - C. A class used only for inheritance
   - D. A class that must be declared static

10. (1 point) What is the role of a friend class in C++?
   - A. To inherit all members from another class
   - **B. To access private and protected members of another class**
   - C. To override protected methods
   - D. To define virtual functions in another class

11. (1 point) What is required for a function to be considered **overloaded**?
   - **A. Same name, different number or type of parameters**
   - B. Same name and same parameters in a derived class
   - C. Different return types only
   - D. Virtual keyword must be used

12. (1 point) Which of the following is true about **function overriding** in C++?
   - A. It requires functions with different names
   - B. It only occurs between unrelated classes
   - **C. It must involve a base class function marked `virtual`**
   - D. It can only happen inside the same class

13. (1 point) When does **function overloading** get resolved?
   - A. At runtime
   - B. At link-time
   - **C. At compile-time**
   - D. During memory allocation

14. (1 point) What does the snippet below demonstrate?

```
class Animal {
public:
    virtual void speak();
    void speak(string);
};

class Dog : public Animal {
public:
    void speak();
};
```

   - A. Function overloading
   - B. Function overriding
   - **C. Function overloading & overriding**
   - D. Function overriding & inheritance

15. (1 point) Can you overload **constructors** in C++?
   - A. No, constructors cannot be overloaded
   - B. Yes, if the return types are different
   - **C. Yes, based on number and type of parameters**
   - D. Only for derived classes

16. (1 point) An object is a(n) _______________ of a class that resides in _______________ and has _______________.
   A. instance; memory; attributes and behavior (state)
   B. memory; on disk; methods only
   C. copy; cache; constructors
   D. method; program; parameters and types

17. (1 point) Why is overriding a public pure virtual function as `protected` potentially problematic?
   A. The compiler will issue a warning
   B. It violates the intent of the public interface, breaking polymorphism
   C. It creates a memory leak
   D. It results in object slicing

18. (1 point) What happens if a derived class overrides a **public pure virtual** function but makes it `protected`?
   A. The override fails to compile
   B. The override is legal, but external code cannot call the function via a base class pointer
   C. The function becomes public again automatically
   D. The base class becomes abstract again

19. (1 point) Which of the following best illustrates the concept of **abstraction**?
   A. Making certain member functions `private` to prevent external access
   B. Hiding whether a class uses an array or a linked list internally
   C. Combining data and methods in a single unit
   D. Protecting internal data from modification

20. (1 point) **Encapsulation** is primarily concerned with:
   A. Protecting how data and behavior are implemented and accessed
   B. Designing systems at a high conceptual level
   C. Describing what a system should do without implementation details
   D. Eliminating redundancy across multiple classes

21. (1 point) Which of the following statements is **true**?
   A. Abstraction hides data; Encapsulation hides complexity
   B. Encapsulation hides data; Abstraction hides complexity
   C. Both abstraction and encapsulation expose implementation details
   D. Encapsulation is only possible in procedural programming

22. (1 point) Why might a base class declare certain data members as `private`?
   A. To prevent derived classes from modifying internal details and breaking invariants
   B. To make them visible only to friend functions
   C. To allow derived classes to override them
   D. To automatically make them read-only

23. (1 point) What is the risk of changing a base class's private member to `protected`?
   A. Derived classes can accidentally (or intentionally) corrupt the base class's internal state
   B. The class will no longer compile
   C. The base class can no longer use its own member functions
   D. The derived class loses access to inherited constructors

24. (1 point) What is the only technical difference between a `struct` and a `class` in C++?

   A. A struct cannot have member functions

   B. A struct defaults to public access, while a class defaults to private

   C. A class can have constructors, but a struct cannot

   D. Structs can only be used with primitive data types

25. (1 point) What does it mean to **instantiate a class**?

   A. To define it as an abstract base class

   B. To create an object from that class

   C. To inherit from the class in a derived class

   D. To overload one of its methods

26. (1 point) Does the `friend` keyword fit cleanly into the object-oriented programming (OOP) paradigm?

   A. Not entirely — it breaks encapsulation by allowing external access to private members

   B. Yes — it reinforces inheritance and abstraction

   C. Yes — because friends must always be member functions

   D. No — because it causes runtime polymorphism errors

27. (1 point) Which access level can be bypassed using the `friend` keyword in C++?

   A. private

   B. public

   C. protected

   D. internal

28. (1 point) When deciding between **composition** and **inheritance** in C++, which of the following best describes the typical guideline?

   A. Use inheritance when you want to hide the implementation details of helper classes

   B. Use composition when one class "has a" another; use inheritance when one class "is a" specialized version of another

   C. Use composition when you need access to protected members

   D. Use inheritance only if both classes are unrelated

29. (1 point) In the following class hierarchy, what is the relationship between the two `print()` methods?

```cpp
class Character {
protected:
    string name;
public:
    void print() {
        cout << name << endl;
    }
};

class Wizard : public Character {
public:
    void print() {
        cout << name << " is a Wizard!" << endl;
    }
};
```

   A. The `Wizard` class is overriding the `print()` method from `Character`

   B. The `Wizard` class is overloading the `print()` method

   C. The two methods are unrelated due to access level

   D. This is an example of constructor overloading

30. (1 point) Is the `print()` method in the `Character` class an example of overloading, overriding, or neither?
    - A. Overloading, because it has a simple parameter list
    - B. Overriding, because it will be redefined in derived classes
    - **C. Neither — it is a regular method with no overloading or overriding in the base class**
    - D. Both — it can act as both overloaded and overridden depending on use

31. (1 point) How can we make a class **abstract** in C++?
    - A. By declaring all its methods as `protected`
    - B. By giving it a constructor with default arguments
    - **C. By including at least one pure virtual function (e.g., `virtual void f() = 0;`)**
    - D. By inheriting from another abstract class

32. (1 point) What is the primary purpose of an **abstract class** in C++?
    - A. To serve as a container for global constants
    - **B. To define a common interface for derived classes without providing full implementation**
    - C. To reduce memory usage in object creation
    - D. To support function overloading

33. (1 point) How many objects can be created directly from an abstract class in C++?
    - A. As many as needed, depending on constructor parameters
    - **B. Zero — abstract classes cannot be instantiated**
    - C. Only one, due to the singleton pattern
    - D. One per derived class

34. (1 point) What is a key difference between an **abstract class** and an **interface** in C++ (noting that C++ doesn't have built-in interfaces)?
    - **A. An abstract class can have both implemented and unimplemented methods; an interface typically has only pure virtual methods**
    - B. Interfaces allow multiple inheritance, while abstract classes do not
    - C. Abstract classes cannot be inherited
    - D. Interfaces support constructors, but abstract classes do not

35. (1 point) What does it mean when we say, "The use of an object of one class in the definition of another class"? For example: a `Line` uses two `Point` objects in its definition.
    - **A. It's an example of composition — a "has-a" relationship**
    - B. It's inheritance — a "is-a" relationship
    - C. It's function overloading
    - D. It's data abstraction

36. (1 point) In an oversimplified way, encapsulation deals with hiding things and ___________ deals with exposing things.
    - A. Polymorphism
    - **B. Abstraction**
    - C. Inheritance
    - D. Virtualization

37. (1 point) An abstract method in C++ is a regular method declared as a pure virtual function (e.g., `= 0`). In other object-oriented languages, this is commonly referred to as a(n):
    - **A. Interface method**
    - B. Static method
    - C. Concrete method
    - D. Friend function

38. (1 point) If you define a base class method as `public` and `pure virtual`, does this force derived classes to keep it `public`?

A. Yes — public access must be preserved or polymorphism will fail

B. No — access level can be changed in derived classes, though it may break the public interface

C. Only if the base class is also abstract

D. Only if `virtual` is omitted

39. (1 point) Do we typically choose inheritance over composition in object-oriented design?

A. Yes — inheritance is always more flexible and reusable

B. No — composition is often preferred because it allows for more modular and maintainable code

C. Yes — composition only works with static members

D. No — inheritance is illegal in modern C++

40. (1 point) How can you rewrite the `print()` function below to make it a **pure virtual method**?

```
class Character {
protected:
    int name;
public:
    void print() {
        cout << name << endl;
    }
};
```

A.
```
virtual void print() { cout << name << endl; }
```

B.
```
virtual void print() = 0;
```

C.
```
void virtual print() = 0 { cout << name << endl; }
```

D.
```
pure virtual void print();
```

```
#include <iostream>
#include <string>

class Logger {
private:
    std::string logFile;  // Only the Logger should know about this!

protected:
    void writeLog(const std::string& message) {
        std::cout << "[LOG] " << message << std::endl;
    }

public:
    Logger(const std::string& filename) : logFile(filename) {}
    virtual ~Logger() = default;

    void log(const std::string& message) {
        // internal housekeeping logic
        writeLog(message);
    }
};

class NetworkLogger : public Logger {
public:
    NetworkLogger(const std::string& filename) : Logger(filename) {}
    void logConnection(const std::string& ip) {
        // Can't access logFile directly | that's intentional
        writeLog("Connection from: " + ip);
    }
};
```

41. (1 point) In the `Logger` example, why is the member `logFile` declared `private` instead of `protected`?
    - **A. Because derived classes like `NetworkLogger` or `EvilLogger` should not be able to change the file destination**
    - B. Because private members are faster to access than protected ones
    - C. Because protected members cannot be used in virtual functions
    - D. Because private variables are inherited automatically