Name: ____________________________

Question	Points	Score
1	34	
2	8	
3	8	
4	8	
5	12	
6	10	
7	4	
8	8	
9	15	
10	15	
Total:	122	

2143 OOP Exam 2

October 23, 2024

Multiple Choice

1. (34 points) Answer the multiple choice questions below. Read them carefully.

   (1) What is the difference between a *class* and an *object*?
   - **A. state**
   - B. objects don't have class
   - C. classes don't have objects
   - D. classes nor objects have state

   (2) Which statement below is correct when discussing *virtual functions* and *pure virtual functions*?
   - A. You can't declare an instance of a class with a virtual function in it.
   - **B. You can't declare an instance of a class with a pure virtual function in it.**
   - C. You can make an interface with a class that has a virtual function in it.
   - D. You can't make an interface with a class that has a pure virtual function in it.
   - E. None of these are correct

   (3) What is a potential drawback of using the friend keyword?
   - A. Potential security risk
   - B. Your privates are exposed
   - C. It breaks encapsulation
   - **D. All of the above**

   (4) Like private members, protected members are inaccessible outside of the class. However, they can be accessed by?
   - A. Super Classes
   - B. Deranged Classes
   - C. Sibling Classes
   - D. Parent Classes
   - **E. None of the above**

   (5) What type of data member can be shared by all instances of its class?
   - **A. static members**
   - B. dynamic members
   - C. public members
   - D. private members

   (6) A constructor is executed when ____**B&C**____?
   - A. You compile your program.
   - B. You use the new operator.
   - C. You declare an instance of a class
   - **D. B & C**
   - E. A & C

   (7) What makes something an object? (as opposed to a class)
   - A. In memory
   - B. Instantiated
   - C. Declared
   - **D. All of the above**

(8) A class that has all of its methods implemented, and can be instantiated is know as a(n): **A Concrete Class** class.

A. A Completed Class
B. A Cement Class
C. A Concrete Class
D. A Constructed Class

(9) What is the one thing that is necessary for Run Time polymorphism?

A. Deleting memory
B. Pointers
C. Dynamic memory
D. All of these play a part

(10) An interface is a C++ class that:

A. With at least one pure virtual method.
B. With no implementation at all.
C. That cannot be instantiated.
D. B & C
E. All of the above

(11) Which of the following is a mechanism of static polymorphism?

A. Operator overloading
B. Function overloading
C. Templates
D. All of the above

```cpp
class Animals {
public:
    virtual void sound() {
        cout << "Playing generic animal sound..." << endl;
    }
};
class Dogs : public Animals {
public:
    void sound() {
        cout << "Dogs bark..." << endl;
    }
};
int main() {
    Animals *a;
    Dogs     d;
    a = &d;
    a->sound();
    return 0;
}
```

(12) The concept portrayed in the previous snippet is known as **Runtime Polymorphism** ? (choose the best answer)

A. Polymorphism
B. Runtime Polymorphism
C. Compile-time Polymorphism
D. Inheritance
E. None of these

(13) What is the output of the previous code snippet?

A. Nothing
B. "Playing generic animal sound..."
C. "Dogs bark..."
D. Both outputs since its a virtual method.
E. None of these

```cpp
class BaseDisplay {
public:
    // Method overloading
    void display() {
        std::cout << "Display with no arguments" << std::endl;
    }

    void display(int i) {
        std::cout << "Display with int: " << i << std::endl;
    }

    void display(double d) {
        std::cout << "Display with double: " << d << std::endl;
    }
};
```

(14) If I wanted to make this an abstract class, at a minimum what would I need to do?

A. Nothing
B. Make one method pure virtual
C. Make two methods pure virtual
D. Make all methods pure virtual
E. None of these

(15) If I wanted to make that same class an interface, at a minimum what would I need to do?

A. Nothing
B. Make one method pure virtual
C. Make two methods pure virtual
D. Make all methods pure virtual
E. None of these

```cpp
#include <iostream>

void print(int i) {
    std::cout << "Printing int: " << i << std::endl;
}

void print(double d) {
    std::cout << "Printing double: " << d << std::endl;
}

void print(const std::string& s) {
    std::cout << "Printing string: " << s << std::endl;
}
```

(16) Would the above snippet error? Can we have functions with the same name outside of a class?

A. No difference
B. Would cause scoping errors.
C. It needs a namespace
D. Just make each function virtual
E. None of these

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

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

(17) The above snippet would error because:

- **A. It wouldn't.**
- B. It would because there would be a name collision since Wizard is a subclass of Character.
- C. It needs a namespace
- D. It needs a scope resolution operator to disambiguate the print calls.
- E. None of these

Vocabulary

#	Word	#	Word	#	Word
1	Derived Class	17	Static Member	33	Hierarchical-Inheritance
2	Polymorphism	18	Constructor	34	Overloading
3	Static Polymorphism	19	Inheritability	35	Pure Polymorphism
4	Dynamic Polymorphism	20	Operator Overloading	36	Friends
5	Virtual Function	21	Composition	37	Interface
6	Pure Virtual Function	22	Instantiation	38	Abstract Base Class
7	Abstract Class	23	Methodization	39	Abstraction
8	Concrete Class	24	Dynamic Memory Allocation	40	Instance-Variable
9	Friend Class	25	New	41	Member-Variable
10	Encapsulation	26	Delete	42	Multilevel-Inheritance
11	Inheritance	27	Friend Function	43	Diamond Problem
12	Multiple Inheritance	28	Static Method	44	Virtual
13	Access Modifiers	29	Virtualizationism	45	Static Member
14	Protected	30	Pure Virtual	46	Class
15	Private	31	Object	47	Overriding
16	Public	32	Class-Variable	48	Method
49	Run-time Polymorphism	50	Encapsulization	51	Destructor
52	Abstractification	53	Classology	54	Pointer
55	Polymorphication	56	Compile-time Polymorphism		

Table 1: Vocabulary Words

2. (8 points) Class Members and Functions
   (A) A special member function that is executed when an object is created. **Constructor**
   (B) A special member function executed when an object is destroyed, used to clean up resources. **Destructor**
   (C) A class member that is shared among all instances of the class. **Static Member**
   (D) A method that belongs to the class rather than any object instance and can be called on the class itself. **Static Member**
3. (8 points) Memory Management and Pointers
   (A) A variable that stores the memory address of another variable or object. **Pointer**
   (B) The process of allocating memory at runtime using pointers (e.g., new and delete in C++). **Dynamic Memory Allocation**

(C) A keyword used to dynamically allocate memory for an object or variable. **New**

(D) A keyword used to deallocate dynamically allocated memory. **Delete**

4. (8 points) Inheritance and Relationships

   (A) A class that is derived from another class. **Derived Class**

   (B) The mechanism by which one class can inherit the properties and methods of another class. **Inheritance**

   (C) A type of inheritance where a class can inherit from more than one base class. **Multiple Inheritance**

   (D) A class that has access to the private and protected members of another class. **Friend Class**

5. (12 points) Polymorphism and Overloading

   (A) The ability of different classes to be treated as instances of the same class through inheritance. **Polymorphism**

   (B) Polymorphism that is resolved during compile time. **Static Polymorphism**

   (C) Polymorphism that is resolved during runtime. **Dynamic Polymorphism**

   (D) The ability to define or alter the behavior of operators (e.g., +, -) for user-defined types. **Operator Overloading**

   (E) A member function that can be overridden in a derived class to provide specific implementation. **Virtual Function**

   (F) A virtual function with no implementation, forcing derived classes to provide an implementation. **Pure Virtual Function**

6. (10 points) Access Modifiers and Encapsulation

   (A) Keywords that define the accessibility of class members (e.g., public, private, protected). **Access Modifiers**

   (B) An access modifier that allows members to be accessed by derived classes and classes within the same package. **Protected**

   (C) An access modifier that restricts access to members to within the same class only. **Private**

   (D) An access modifier that allows members to be accessed from any other code. **Public**

   (E) The bundling of data and methods that operate on the data within one unit, such as a class. **Encapsulation**

7. (4 points) Class Types

   (A) A class that cannot be instantiated and is designed to be inherited by other classes. **Abstract Class**

   (B) A class that can be instantiated and is not abstract. **Concrete Class**

8. (8 points) Other Concepts

   (A) The concept of creating a virtual representation of something, such as hardware or an operating system. **Virtualization**

   (B) The use of an object of one class as a member of another class, indicating a "has-a" relationship. **Composition**

   (C) The process of creating an instance of a class (i.e., an object). **Instantiation**

   (D) A function that is not a member of a class but has access to its private and protected members. **Friend Function**

Inheritance V Composition

9. (15 points) Determine which to use: Inheritance or Composition

   (A) Laptop **IS-A** Computer

   (B) Smartphone **HAS-A** Touchscreen

   (C) Compiler **IS-A** Program

   (D) Car **HAS-A** GPS System

   (E) Python **IS-A** Programming Language

   (F) Web Browser **HAS-A** Address Bar

   (G) User Interface **HAS-A** Button

   (H) A CPU **IS-A** Hardware Component

   (I) Computer **HAS-A** CPU

   (J) A LinkedList **IS-A** Data Structure

   (K) A Dog **IS-A** Animal

   (L) A Car **HAS-A** Engine

   (M) A Car **IS-A** Vehicle

   (N) A Square **IS-A** Shape

   (O) A Line **HAS-A** Point

Inheritance Types

10. (15 points) Label each Ascii art drawing with its appropriate inheritance type.

(A) **Single**

```
    BaseClass
        |
    DerivedClass
```

(B) **Multi-Level**

```
     BaseClass
        |
    DerivedClass1
        |
    DerivedClass2
```

(C) **Hierarchical**

```
    BaseClass
      /   \
Derived1  Derived2
```

(D) **Multiple**

```
  Base1    Base2
     \      /
     Derived
```

(E) **Hybrid**

```
 Base1    Base2
    \      /
    Derived1
       |
    Derived2
```