OOP Sheet

Question(1) True/False Questions:

1. In object-oriented programming, a class is a blueprint for creating objects.
2. Procedural programming uses a bottom-up approach.
3. Constructors can have a return type in C++.
4. Encapsulation in OOP means exposing all the internal details of an object.
5. In C++, a constructor must have the same name as its class.
6. In object-oriented programming, classes are used to create multiple instances known as objects.
7. Encapsulation hides internal data and only exposes selected features.
8. Overloading is possible in object-oriented programming.
9. Constructors can be overloaded with different types or numbers of parameters.
10. Destructors can take parameters.
11. Encapsulation increases the security of the data in a program.
12. A copy constructor is used to create an object by copying another existing object.
13. Static member variables are defined outside their class declaration.
14. If a class doesn't have a copy constructor, the compiler generates a default copy constructor for it.
15. The base class's access specification affects the way the derived class inherits members of the base class.
16. Public members of a private base class become private members of the derived class.
17. Public members of a protected base class become private members of the derived class.
18. Protected members of a public base class become public members of the derived class.
19. The base class destructor is called after the derived class destructor.
20. Arguments are passed to the base class constructor by the derived class constructor.
21. You must declare all private members of a class before the public members.
22. Constructors are never declared with a return data type.
23. Destructors are never declared with a return type.
24. Destructors may take any number of arguments.
25. Private members must be declared before public members.
26. All private members of a class must be declared together.
27. A private member function may be called from a statement outside the class, as long as the statement is in the same program as the class declaration.
28. Programming paradigms differ in steps that compose a computation only.
29. C++ supports only one programming paradigm.
30. C++ supports procedural programming and object-oriented programming.
31. Java supports procedural programming and object-oriented programming.
32. Using a procedural language, the programmer specifies language statements to perform a sequence of algorithmic steps to reach the desired state based upon the concept of the procedure call.
33. Procedural programming provides unrestricted access to data.
34. In procedural programming it is difficult to relate with real objects.

Question(2)Multiple Choice Questions:

1. **Which of the following is NOT a benefit of Object-Oriented Programming (OOP)?**
   - A) Modularity
   - B) Code Reusability
   - C) Complexity increase
   - D) Security
2. **Which access specifier allows access only within the class itself?**
   - A) Public
   - B) Protected
   - C) Private
   - D) External
3. **Which of the following C++ concepts allows objects to be created from classes?**
   - A) Abstraction
   - B) Inheritance
   - C) Instantiation
   - D) Polymorphism
4. **In a class, what operator is used to access an object's members?**
   - A) ->
   - B) .
   - C) ::
   - D) #
5. **What is the correct characteristic of procedural programming?**
   - A) Focus on objects
   - B) Data hiding
   - C) Top-down approach
   - D) Bottom-up approach
6. Which of the following best defines **Instantiation**?
   - A) Creating a new function
   - B) Creating a new object from a class
   - C) Deleting an object
   - D) Hiding data members
7. Which access specifier allows class members to be accessed by subclasses but not by the general public?
   - A) Public
   - B) Private
   - C) Protected
   - D) Final
8. Which is **NOT** a principle of Object-Oriented Programming?
   - A) Encapsulation
   - B) Inheritance
   - C) Compilation
   - D) Polymorphism
9. What will the constructor of a class mainly be used for?
   - A) Create copies
   - B) Delete data members
   - C) Initialize data members
   - D) Overload operators

6. What is a key feature of a constructor in C++?
    - A) It must have a return type.
    - B) It can have any name.
    - C) It initializes objects automatically.
    - D) It cannot be overloaded.
7. In C++, how many destructors can a class have?
    - A) Unlimited
    - B) Exactly one
    - C) Two
    - D) As many as constructors
8. Which of the following statements is true about encapsulation?
    - A) Class attributes should always be public.
    - B) Class attributes should be private and accessed through getters and setters.
    - C) Class attributes should be protected to be accessed globally.
    - D) Private attributes cannot be accessed at all.
9. Which of this best describes a copy constructor?
    - A) Initializes an object using a constant value.
    - B) Creates a duplicate object using another object of the same class.
    - C) Deletes object attributes when memory is reclaimed.
    - D) Changes the private members directly.

10. What happens when you call the destructor manually in C++?
    - A) It resets the object.
    - B) It deallocates dynamic memory and releases resources.
    - C) It calls the constructor again.
    - D) It creates a new object.

11. What does OOP stand for in C++?
1. Object-Oriented Programming
2. Overloaded Operator Programming
3. Ordered Operation Procedure
4. Object Operation Protocol

12. What is the role of a class in C++?
    1. To perform calculations within a program
    2. To define a template or blueprint for creating objects
    3. To handle file operations
    4. To act as an external library
13. How can methods be defined in a C++ class?
    1. Only inside the class definition
    2. Only outside the class definition
    3. Inside or outside the class definition
    4. Only in the main function
14. How do you define a constructor in C++?
    1. By defining a function with the keyword 'constructor'
    2. By declaring it as 'public constructor'

3. Using a function named 'init'
4. Using the class name followed by parentheses

15. What is the role of access specifiers in C++?
    1. To create instances of a class
    2. To initialize variables
    3. To define how class members can be accessed
    4. To define the main function
16. **Select the following which shows the correct constructor.**
    a) **()class_name**
    b) **~ class_name**
    c) **class_name()**
    d) **~ class_name()**
17. **To access data members of a class, which of the following is used?**
    a) **Dot Operator**
    b) **Arrow Operator**
    c) **Dot or Arrow Operator as required**
    d) **Dot, arrow, or direct call**

18. **Total access specifiers in OOPS for C++ are?**
    a) **1**
    b) **2**
    c) **3**
    d) **4**
19. **Total types of constructors in C++ are?**
    a) **1**
    b) **2**
    c) **3**
    d) **4**

20. Which is private member functions access scope?

a) Member functions which can only be used within the class
b) Member functions which can used outside the class
c) Member functions which are accessible in derived class
d) Member functions which can't be accessed inside the class

21. Which among the following is true?

a) The private members can't be accessed by public members of the class
b) The private members can be accessed by public members of the class
c) The private members can be accessed only by the private members of the class
d) The private members can't be accessed by the protected members of the class

22. Which member can never be accessed by inherited classes?

a) Private member function
b) Public member function
c) Protected member function
d) All can be accessed

23. **Which syntax among the following shows that a member is private in a class?**

a) private: functionName(parameters)
b) private(functionName(parameters))
c) private functionName(parameters)
d) private::functionName(parameters)

24. **Which of the following is correct about function overloading in C++?**

- A) Function overloading is based on the return type only.
- B) Function overloading is based on the function name.
- C) Function overloading is based on the number or type of arguments.
- D) Function overloading is not allowed in C++.

25. What will happen if you attempt to overload a function by only changing the return type in C++?

a. A) It will compile successfully.
b. B) It will result in a compile-time error.
c. C) It will cause a runtime error.
d. D) The compiler will automatically resolve the ambiguity.

26. Which of the following function declarations are valid for function overloading?

a. A) `void add(int, float);` and `void add(float, int);`
b. B) `void add(int, float);` and `void add(int);`
c. C) `void add(int, float);` and `void add(float, float);`
d. D) All of the above.

27. Which of the following statements is true about default arguments in overloaded functions?

a. A) You cannot use default arguments in function overloading.
b. B) Default arguments can only be used in non-overloaded functions.
c. C) Default arguments allow the function to be called with fewer parameters than defined.
d. D) Overloaded functions with default arguments are always ambiguous.

28. What happens if you overload a function in C++ but the arguments provided during a function call don't match any of the overloads?

a. A) The program will not compile.
b. B) A runtime error occurs.
c. C) The compiler will choose the most compatible overload automatically.
d. D) The function will be executed with default arguments.

Which of the following is NOT a valid reason to overload a function in C++?

A) To allow a function to accept different types of arguments.

e. B) To allow a function to accept a varying number of arguments.
f. C) To create a function that behaves differently depending on the number or type of arguments.
g. D) To modify the return type of a function without changing the parameters.

29. Which of the following statements is true for function overloading in C++?

a. A) Function overloading requires that all overloaded functions must have the same number of arguments.
b. B) Function overloading requires that all overloaded functions must have the same return type.
c. C) Function overloading is determined by the number or type of arguments in the function signature.
d. D) Function overloading is determined by the order of function calls.

31.What does the following overloaded `+` operator for the class `Point` do?

```
Point operator+(const Point& p) {
    return Point(x + p.x, y + p.y);
}
```

- A) Adds the x and y values of two `Point` objects and returns a new `Point`.
- B) Changes the values of the current object's `x` and `y` by adding them to another `Point`.
- C) Returns the sum of the two `Point` objects without modifying any of them.
- D) Adds two `Point` objects and prints the result.

32. When overloading the `++` operator for a class `Counter`, which of the following is correct for the post-increment operator?

a. A) `Counter operator++(int)` should be used.
b. B) `Counter operator++()` should be used.
c. C) `Counter operator--(int)` should be used.
d. D) Both `Counter operator++(int)` and `Counter operator++()` can be used for post-increment.

33. Which of the following operators cannot be overloaded?

- A) `=`
- B) `+`
- C) `[]`
- D) `new`

34. In the context of overloading operators in C++, what does the `operator+` function return when overloading the addition operator for two objects of a class?

a. A) The result is added to the left operand and the left operand is modified.
b. B) The result is added to the right operand and the right operand is modified.
c. C) A new object is returned with the result of the addition.
d. D) The function does not return any value.

35. Which of the following is true when overloading the `[]` operator in a class?

a. A) The operator can only be overloaded as a member function.
b. B) The operator can be overloaded as both a member and non-member function.
c. C) It must return a reference to an element in the object.
d. D) Both B and C.

Programs:

Question 1:

a) Define a class BankAccount with the following:

- Data members: accountNumber, accountHolder, and balance (all private).
- Member functions:
  - openAccount() – Reads and sets account details.
  - displayAccount() – Displays account info.
  - deposit(amount) – Adds money to balance.
  - withdraw(amount) – Subtracts money (if sufficient).
  - transferTo(anotherAccount, amount) – Transfers money between two accounts.

b) Write a program that declares two objects of class BankAccount, reads their data from the keyboard using openAccount(), then performs deposit and withdrawal operations on each object, then transfers money from the first account to the second using transferTo(), and displays both accounts' details after each operation.

Question 2:

Create a class Distance that reads a distance in centimeters and converts it into feet, inches.

Requirements:

- Class **Distance** with one private attribute: double cm.
- A **constructor** to read a value in centimeters.
- A **destructor** to display a message when an object is destroyed.
- A method **convert ()** to:
  - Calculate and print distance in feet (1 foot = 30.48 cm).
  - Calculate inches (1 inch = 2.54 cm).

Sample Expected Output:

```
Enter distance in cm: 187
Distance is:
Feet: 6
Inches: 1
Destructor called: Object is destroyed.
```

Question 3:

(a) Consider the class Base shown in the listing below. Derive two classes Derived1 and Derived2 from Base, and in each case, define the member function display to output the name of the class.

```
class Base {
public:
virtual void display()
{
cout << "Base\n";
}
};
```

(b) Create an object of each of the types Base, Derived1, and Derived2, and invoke its display member function.

© Create pointers of type Base* that are initialized to point to the preceding Base, Derived1, and Derived2 objects, and call display through each of these pointers

---

Question 4:

a) Suppose that we want a class that can be used to represent people at a university (e.g., faculty, staff, undergraduate students, and graduate students). For all types of people, we want to record their name and university ID. For specific types of people, we want to record some additional information. In the case of faculty, we want to record their rank (i.e., Assistant Professor, Associate Professor, or Full Professor). In the case of undergraduate students, we want to record their year of study (i.e., 1, 2, 3, or 4). In the case of graduate students, we want to record their supervisor's university ID. Suggest a class hierarchy that might be used to represent the above collection of people. You do not need to fully implement the class.

Question 5:

Part A:

Define a base class ***Person***, and two derived classes: ***Professor*** and ***Student***.

- The base class **Person** has two data members: **name** (a string) and **id** (an int), a ***constructor***, and a member function ***displayPerson()*** to display the person name and id.
- The derived class **Student** has a data member **nPapers** (an int represents the no. of research papers published by the student and his/her supervisor), a ***constructor***, and a member function: ***papers()*** that returns the no. of papers.
- The derived class **Professor** has two data members: **StudentsList** (an array of **Student** objects that holds the students supervised by the professor) and **nStudents** (an int represents the no. of those students), a ***constructor***, and three member functions: ***registerStudent()*** that accepts a **Student** object and enters it in the array **StudentsList**, ***total()*** that calculates and returns the total no. of papers published by the professor and his students, and ***displayStudents()*** that displays the data of the students supervised by the professor.

Part B:

Write a main program that reads a professor's name and id, and creates a **Professor** object for that professor. Then, the program reads a series of students' names, ids, and no. of papers, until the name XXX is entered. For each student, a **Student** object is created and entered to the array **StudentsList** of the **Professor** object by registering that student with the professor. After all students have been entered, the data of the students

supervised by the professor are displayed, then the total no. of papers published by the professor and his students is calculated and displayed.

Question 6)

a) Define a class ***fraction*** that has two data members: ***num*** and ***denom***, (both of type **int**), which represent the fraction's numerator and denominator, respectively. It has the member function ***getFract()*** that reads the ***num*** and ***denom*** of a ***fraction*** object from the keyboard in fractional form (i.e. num/denom), the member function ***showFract()*** that displays a ***fraction*** object in fractional form, and the member functions: ***addFract()*** , ***subFract()***, ***mulFract()***, ***divFract()***, for the performing addition, subtraction, multiplication and division operations on any two objects of class ***fraction***, respectively.

b) Write a program that declares two objects of class ***fraction***, reads their data from the keyboard, then perform the four operations on them, and displays the result in each case.

Notes:

(1) The addition of the two fractions a/b and c/d can be performed by the formula:

$$\frac{a}{b}+\frac{c}{d}=\frac{a*d+b*c}{b*d}$$

(2) The subtraction of the two fractions a/b and c/d can be performed by the formula:

$$\frac{a}{b}-\frac{c}{d}=\frac{a*d-b*c}{b*d}$$

Question 7:

1. **Create a base class *Circle* that has 4data members, *x*, *y*, *r* and *area*, representing the coordinates of the center, the radius, and the area of a circle, and has a constructor that initializes the data members of *Circle* objects with given values. It also has 2 member functions: (1) *show_data()* to display the values of the center coordinates and radius of a *Circle* object, and (2) *getArea()* that calculates the area of a *Circle* object by using the formula πr^2, where π=3.14.**

2. Create a child class ***Cylinder*** that has an additional data member ***h*** representing the cylinder height**,** and has a constructor that initializes the data members of the ***Cylinder*** objects with given values. It redefines the member function ***show_data()*** to display the values of the data members of the ***Cylinder*** object, and redefines the member function ***getArea()*** to calculate the surface area of a ***Cylinder*** object by using the formula: **$(2 \times \pi \times r^2) + (2 \times \pi \times r \times h)$**

.

3. Write a program that creates an array of 5 pointers to **Circle**. In a loop, gets from the user data about a circle or a cylinder, and use **new** to create an object of type **Circle** or ***Cylinder*** to hold the data, and then put the pointer to the created object in the array. When the user has finished entering the data for all figures, display the data for all figures entered.

4. Run the program with the following data:

Input Data: (C = Circle, L = Cylinder)

Type	x	y	radius	height
C	2.5	4.6	10	
C	5.25	3.8	12	
L	7.8	4.5	8	10
C	9.5	6.4	20	
L	3.9	8.25	15	12

Question 8:

(a) Define a class *counter*, which has three data members: *count* (of type int) that represents the value of the counter, *overflow* (of type bool) that is set to *true* if an overflow occurs, and *maxcnt* (of type int) that represents the maximum value of the counter. It has a constructor *counter(int max_count)*, that creates a counter with a given maximum count and an initial value 0, and sets *overflow* to *false*, and it has 4 member functions: *increment()*, which increases the current count by 1; but if the current count equals the maximum count, the *count* is set to zero, and *overflow* to true; *reset()*, which sets the counter to zero and overflow to false; *get_count()*, which returns the current count, and *check_for_overflow()*, which returns true if an overflow has occurred and false if it has not.

(b) Write a main program that reads a sentence, and counts the frequency of each vowel (a, e, i, o, u) in it, then displays these frequencies. The program should use a separate counter for each vowel, and it should display an error message if any of the counters overflows.

Question 10)

(a) Define a base class **Employee** that has two data members ***name*** and ***idno***. From this class derive two classes: **SalaryEmployee**, which adds a data member ***salary***; and **TempEmployee**, which adds two data members: ***hourlyPay*** and ***hoursWorked***. Each of the three classes should

have two member functions: ***Getdata()*** to get its data from the user, and ***Showdata()*** to display its data.

(b) Write a program that creates an array of 10 pointers to **Employee**. In a loop, gets from the user data about a salary employee or a temporary employee, and use **new** to create an object of type **SalaryEmployee** or **TempEmployee** to hold the data, and then put the pointer to the created object in the array. When the user has finished entering the data for all employees, display the data for all employees entered.

Question 11)

a- Design a class named `Triangle` to represent a geometric triangle. The class should include:

- **Three integer properties** to store the lengths of the triangle's sides.
- A method to **verify if the given sides form a valid triangle** using the triangle inequality rule:
  - The sum of any two sides must be greater than the third side.
- A method to **classify the triangle** as one of the following:
  - `Equilateral` (all three sides are equal),
  - `Isosceles` (two sides are equal), or
  - `Scalene` (all sides are different).
- A method to **output the triangle's side lengths and classification**, or a message indicating the sides do not form a triangle.

B- write a C++ program that:

1. Prompts the user to input **three integers** representing the sides of a potential triangle.
2. Creates an object of the `Triangle` class using the input values.
3. Displays:
   - The side lengths entered by the user,
   - Whether the sides form a valid triangle, and if so,
   - The **type** of the triangle (equilateral, isosceles, or scalene),
   - Otherwise, display a message indicating the input does **not** form a valid triangle.

Question (12):

a) Design a class named `MonthInfo` that represents a calendar month and provides information about its number of days. The class should include:

- Two properties:
  - `month` (an integer between 1 and 12)
  - `year` (a four-digit integer)
  - also have:

1- A method to **determine whether the year is a leap year**, using the standard rule:

A year is a **leap year** if:

- It is **divisible by 4** (i.e., `year % 4 == 0`) **AND** It is **NOT divisible by 100** → (i.e., `year % 100 != 0`).
- **UNLESS** It **is divisible by 400** → (i.e., `year % 400 == 0`)

2- A method that returns the **number of days in the specified month**, taking leap years into account when the month is February.

3- A method to **return the month name** (e.g., 1 → "January", 2 → "February", etc.).
4- A method to format and return a string like:

- `"February 2000 has 29 days."` or
- `"March 2005 has 31 days."`

B) Implement a C++ program that:

1. Prompts the user to enter:
   - A **month number** (1–12)
   - A **year**
2. Creates an instance of the `MonthInfo` class using the entered values.
3. Displays:
   - The **name of the month**
   - The **year**
   - The **correct number of days** in that month.

The output should be user-friendly and formatted like this:

```
February 2000 has 29 days.
```

Question 3: what the output of the following code is?

a)

```
#include <iostream>
using namespace std;

class Animal {
public:
   virtual void speak() {
      cout << "Animal speaks" << endl;
   }
};
class Dog : public Animal {
public:
   void speak() override {
      cout << "Dog barks" << endl;
   }
};
int main() {
   Animal* a = new Dog();
    a->speak();
}
```

b)

```
#include <iostream>
using namespace std;
class Shape {
public:
   void draw() {
      cout << "Drawing a shape" << endl;
   }
};
class Circle : public Shape {
public:
   void draw() {
      cout << "Drawing a circle" << endl;
   }
};
int main() {
   Shape* s;
   Circle c;
```

```
    s = &c;
    s->draw();
    return 0;
}
```

c)

```
#include <iostream>
using namespace std;

class Student {
private:
    int id;
    string name;

public:
    void setData(int i, string n) {
        id = i;
        name = n;
    }
    void showData() {
        cout << "ID: " << id << ", Name: " << name << endl;
    }
};

int main() {
    Student s1;
    s1.setData(101, "Ali");
    s1.showData();
    return 0;
}
```

d)

```
#include <iostream>
using namespace std;

class Point {
private:
    int x, y;

public:
    Point() {
        x = 0; y = 0;
    }

    Point(int a, int b) {
        x = a; y = b;
    }

    void display() {
        cout << "Point: (" << x << "," << y << ")" << endl;
    }
};

int main() {
    Point p1;
    Point p2(5, 10);
    p1.display();
    p2.display();
    return 0;
}
```

e)

```
#include <iostream>
using namespace std;

class Shape {
public:
    virtual void draw() = 0; // pure virtual function
};

class Square : public Shape {
public:
    void draw() override {
        cout << "Drawing Square" << endl;
    }
};

int main() {
```

Square s; **s.draw();** **return 0;** **}**	

Show the output of the following programs:

1)

```
class Fract {
  private:
    int a, b;
  public:
    Fract(int x=0, int y=1)
    {  a = x; b = y; }
Fract operator+(const Fract& f) {
        Fract temp;
        temp.a = a * f.b + b * f.a;
        temp.b = b * f.b;
        return temp;
    }
    Fract operator*(const Fract& f) {
        Fract temp;
        temp.a = a * f.a;
        temp.b = b * f.b;
        return temp;
        }
Fract& operator=(const Fract& f) {
        a = f.a;
        b = f.b;
        return *this;
    }
    void Display() {
        cout << "\n" << "("<< a<< "/"
             << b << ")";
    }
};
void main() {
```

```
    Fract f1(3,2);
    Fract f2(8,3);
    Fract f3, f4;
    f1.Display();       f2.Display();
    f3.Display();       f4.Display();
    f3 = f1 + f2;       f3.Display();
    f4 = f1 * f2;       f4.Display();
}
```

2)

```
class Point {
   protected:
      int x, y;
   public:
     Point(int xx, int yy)
     { x = xx;  y = yy; }
     void display()
     {cout << "\n(" << x << "," << y << ")"; }
     virtual float getArea()
     { return 0.0; }
};

class Circle : public Point {
   private:
     float radius;
   public:
     Circle(int xx, int yy, float r): Point(xx, yy)
     { radius = r; }

     void display()
     { cout << "\n(" << x << "," << y
            << "," << radius << ")"; }
     float getArea()
     { return 3.14*radius*radius; }
};

void main () {
  Point* List[4];
  List [0]=new Circle (2, 6, 10.0);
  List [1]=new Point (4, 5);
  List [2]=new Circle (3, 4, 20.0);
  List [3]=new Point(0, 0);
  for (int i=0; i<4;i++) {
    List [i]->display();
    cout << "area=" << List [i]-> getArea();
   }
```

}

3)

```
class weight2 {

private:

int pound;  float ounces;

   public:

weight2()  { pound = 0;  ounces = 0.0; }

weight2(int p, float on)

{ pound = p;  ounces = on; }
```

```
      void display()
      { cout << "\npound=" << pound
             << "\tounces=" << ounces; }
       friend class weight1;
};   // End of class weight2
class weight1 {
   private:
     int kg;  float grams;
   public:
      weight1()   { kg = 0;  grams = 0.0; }
      weight1(int k, float g)
      { kg = k;  grams = g; }
      weight1(weight2 w) {
        float poundValue =
                w.pound + w.ounces/16;
        float kgValue = poundValue * 0.45;
        kg = (int) kgValue;
        grams = (kgValue - kg) * 1000;
       }
       void display()
       { cout << "\nkg=" << kg
              << "\tgrams=" << grams; }
        weighnt1 operator+ (weight1 w) {
          weight1 temp;
          temp.kg = kg + w.kg;
          temp.grams = grams + w.grams;
          return temp;
        }
        operator weight2() {
          float kgValue = kg + grams/1000;
          float poundValue = kgValue * 2.2;
          int pnd = (int) poundValue;
          float oncs = (poundValue - pnd) * 16;
```

```
            return weight2(pnd,oncs);
           }
};  // End of class weight1
void main() {
   weight2 w2(15,8);
   weight1 w1 = w2;
   w2.display();
   w1.display();
   weight1 wt1(10, 20);
    weight1 wt2(15, 80);
   weight1 wt3;
   wt3 = wt1 + wt2;
    wt3.display();
   weight2 wt4;
   wt4 = wt3;
  wt4.display();  }
```