SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 of the UGC Act, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

School of Computing
Second CIA Examination – Oct 2023
Course Code: CSE213
Course Name: Object Oriented Programming
Duration: 90 minutes   Max Marks: 50

PART A (2x10=20)

Answer all the questions

1. **What is meant by encapsulation?**
Encapsulation is one of the fundamental principles of object-oriented programming (OOP) and refers to the bundling of data (attributes or properties) and the methods (functions or procedures) that operate on that data into a single unit known as a class.
2. **Define class and object with suitable example.**
A class in object-oriented programming (OOP) is a blueprint or template for creating objects.
3. **How Code reusability is achieved in OOP?**
Inheritance, polymorphism, template.
4. **What is the need of Virtual inheritance?**
To solve diamond problem.
5. **Write short note on Access Specifiers.**
Public-> Members declared as public are accessible from anywhere in the program, Private-> Members declared as private are not accessible from outside the class. and Protected-> Members declared as protected are similar to private members in that they are not accessible from outside the class. However, they can be accessed by derived classes.

1. **What is meant by Dynamic Memory Allocation?**
Memory allocation during program runtime: New and Delete operator.

6. **Predict the output**

```
 class MyClass {
public:
static int staticVar;
const int constVar;
MyClass(int value) : constVar(value) {}
void displayValues() {
staticVar++;
cout<<"static var="<<staticVar<<" constant
var="<<constVar<<endl;
 } };
int MyClass::staticVar = 0;
int main() {
MyClass obj1(10);
MyClass obj2(20);
obj1.displayValues();
obj2.displayValues();
return 0; }
```

Output:

```
static var=1 constant var=10
static var=2 constant var=20
```

7. **What is an abstract class?**
An abstract class in object-oriented programming is a class that cannot be instantiated on its own but serves as a blueprint for other classes. It contain pure virtual function.
8. **Compare and contrast static binding and late binding.**
Static: Compile time polymorphism, Achieved by Method Hiding
Late binding: Run time polymorphism, Achieved by Method overriding
9. **What are templates in C++? How do they facilitate generic programming?**
Function template-> generic functions that can work with different data types., Class template -> generic classes that can work with different data types. It facilitate code reusability.

PART B (3x10=30)

Answer All the Three Questions

10. **Create a class called Book with the following attributes: title, author, ISBN, publicationYear, and price. Implement necessary constructors and member functions to set and display these attributes. In the main() function, create an array of 3 Book objects, populate them with information, and find and display the details of the book with the highest price using the > operator (as a friend function for comparison).**

Answer:

```
#include <iostream>
#include <string>
class Book {
private:
    std::string title;
    std::string author;
    std::string ISBN;
    int publicationYear;
    double price;
public:
    // Constructors
    Book() {}
    Book(const std::string& t, const std::string& a, const std::string&
isbn, int year, double p)
        : title(t), author(a), ISBN(isbn), publicationYear(year), price(p)
{}
    // Member functions to set and display attributes
    void setAttributes(const std::string& t, const std::string& a, const
std::string& isbn, int year, double p) {
        title = t;
        author = a;
        ISBN = isbn;
        publicationYear = year;
        price = p; }
    void display() const {
        std::cout << "Title: " << title << std::endl;
        std::cout << "Author: " << author << std::endl;
        std::cout << "ISBN: " << ISBN << std::endl;
        std::cout << "Publication Year: " << publicationYear <<
std::endl;
        std::cout << "Price: $" << price << std::endl;
    }
    // Friend function to compare book prices
    friend bool operator>(const Book& book1, const Book& book2) {
        return book1.price > book2.price;
    }
};

int main() {
    Book books[3];
    // Populate book information
    books[0].setAttributes("Book A", "Author A", "ISBN001", 2020,
29.99);
    books[1].setAttributes("Book B", "Author B", "ISBN002", 2018,
24.95);
    books[2].setAttributes("Book C", "Author C", "ISBN003", 2022,
34.50);

    // Find the book with the highest price
    Book highestPriceBook = books[0];
    for (int i = 1; i < 3; ++i) {
        if (books[i] > highestPriceBook) {
            highestPriceBook = books[i];
        }
    }
    std::cout << "Book with the highest price:" << std::endl;
    highestPriceBook.display();
    return 0;
}
```

11. **Create a base class SportsTeam with teamName, coachName, and city attributes. Include a virtual function printAdditionalInfo().**
    - **Create derived classes SoccerTeam, BasketballTeam, and TennisTeam. Override printAdditionalInfo() in each class and add specialized attributes: numPlayers, teamCaptain, and rank.**
    - **Implement constructors to initialize attributes.**

    Answer:

```
#include <iostream>
#include <string>
class SportsTeam {
protected:
    std::string teamName;
    std::string coachName;
    std::string city;

public:
    SportsTeam(const std::string& team, const std::string&
coach, const std::string& teamCity)
        : teamName(team), coachName(coach), city(teamCity) {}
    virtual void printAdditionalInfo() {
        std::cout << "Team: " << teamName << std::endl;
        std::cout << "Coach: " << coachName << std::endl;
        std::cout << "City: " << city << std::endl;
    }
};

class SoccerTeam : public SportsTeam {
private:
    int numPlayers;
    std::string teamCaptain;

public:
    SoccerTeam(const std::string& team, const std::string&
coach, const std::string& teamCity, int players, const
std::string& captain)
        : SportsTeam(team, coach, teamCity),
numPlayers(players), teamCaptain(captain) {}

    void printAdditionalInfo() override {
        SportsTeam::printAdditionalInfo();
        std::cout << "Number of Players: " << numPlayers <<
std::endl;
        std::cout << "Team Captain: " << teamCaptain <<
std::endl;
    }
};

class BasketballTeam : public SportsTeam {
private:
    int numPlayers;
    int rank;

public:
    BasketballTeam(const std::string& team, const std::string&
coach, const std::string& teamCity, int players, int teamRank)
        : SportsTeam(team, coach, teamCity),
numPlayers(players), rank(teamRank) {}

    void printAdditionalInfo() override {
        SportsTeam::printAdditionalInfo();
        std::cout << "Number of Players: " << numPlayers <<
std::endl;
        std::cout << "Rank: " << rank << std::endl;
    }
};

class TennisTeam : public SportsTeam {
private:
    int rank;

public:
    TennisTeam(const std::string& team, const std::string&
coach, const std::string& teamCity, int teamRank)
        : SportsTeam(team, coach, teamCity), rank(teamRank) {}

    void printAdditionalInfo() override {
        SportsTeam::printAdditionalInfo();
        std::cout << "Rank: " << rank << std::endl;
    }
};
int main() {
    SoccerTeam soccer("Soccer Team A", "Coach A", "City X",
22, "Player Z");
    BasketballTeam basketball("Basketball Team B", "Coach B",
"City Y", 15, 2);
    TennisTeam tennis("Tennis Team C", "Coach C", "City Z",
5);
    std::cout << "Soccer Team Info:" << std::endl;
    soccer.printAdditionalInfo();
    std::cout << "\nBasketball Team Info:" << std::endl;
    basketball.printAdditionalInfo();
    std::cout << "\nTennis Team Info:" << std::endl;
    tennis.printAdditionalInfo();

    return 0;
}
```

12. **Write a C++ program that defines a template function findMax to find the maximum of two values of any data type. Additionally, create a template specialization for the char data type to find the maximum character in a case-insensitive manner. In the main function, demonstrate the use of the findMax function with the following test cases:**
    - **Find the maximum of two integers.**
    - **Find the maximum of two doubles.**
    - **Find the maximum of two characters while ignoring case sensitivity.**

Answer:

```
#include <iostream>
#include <cctype>
template <typename T>
T findMax(const T &a, const T &b) {
    return (a > b) ? a : b;
}
template <>
char findMax(const char &a, const char &b) {
    char ca = std::tolower(a);
    char cb = std::tolower(b);
    return (ca > cb) ? a : b;
}
int main() {
    // Test the findMax function with different data types and
cases.
    // Find the maximum of two integers.
    int intA = 10, intB = 20;
    int maxInt = findMax(intA, intB);
    std::cout << "Max of " << intA << " and " << intB << " is: "
<< maxInt << std::endl;
    // Find the maximum of two doubles.
    double doubleA = 12.34, doubleB = 45.67;
    double maxDouble = findMax(doubleA, doubleB);
    std::cout << "Max of " << doubleA << " and " << doubleB <<
" is: " << maxDouble << std::endl;
    // Find the maximum of two characters while ignoring case
sensitivity.
    char charA = 'A', charB = 'b';
    char maxChar = findMax(charA, charB);
    std::cout << "Max of " << charA << " and " << charB << "
(case-insensitive) is: " << maxChar << std::endl;

    return 0;
}
```