Practice Midterm Fall 2024 - SOLUUTIONS

Notes:

- This document is provided as additional exercises for the midterm exam preparation, and it should not be considered as a document to indicate the level of complexity, size, or format of upcoming midterm exam. **Solutions will not be posted on the D2L**
- **You can ignore question related to Design Patterns, as this subject will not be included in this midterm.**

Question 1: Drawing UML Class Diagram

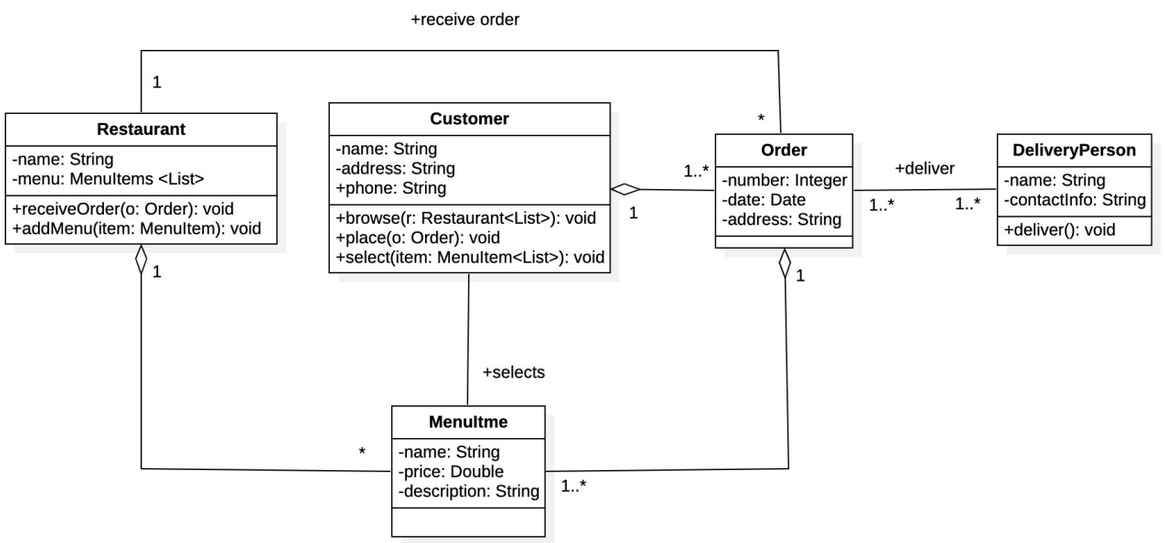
You are designing a **food delivery application** that connects restaurants with customers. The system should be capable of handling the following entities:

Restaurant: Each restaurant has a name, a list of available menu items, and the ability to receive orders from customers.
Menu Item: Each menu item has a name, price, and description. It belongs to a specific restaurant.
Customer: Customers have a name, address, and phone number. They can browse restaurants, select items from the menu, and place orders.
Order: Each order is associated with one customer and one or more menu items. It has an order number, order date, delivery address, and total amount. An order can be placed for multiple menu items from a single restaurant.
Delivery Person: Each delivery person has a name, contact information, and the ability to pick up orders from restaurants and deliver them to customers. They are assigned specific orders for delivery.
Task: Draw a UML class diagram to model this food delivery application. Your diagram should include:

- The classes involved (with attributes and methods).
- Relationships between the classes (associations, inheritance, etc.).
- Any multiplicities (e.g., a Customer can place many Orders, an Order can have multiple Menu Items).
- Access modifiers for attributes and methods (+ for public, - for private).

Make sure to represent relationships such as:

- **Aggregation** or **composition** where appropriate (e.g., a Restaurant contains many Menu Items).
- **Associations** between Order, Customer, Menu Item, and Delivery Person.
- **Inheritance** if applicable (e.g., if you think there's a need for subclasses).



Question 2 – Aggregation and Composition in C++

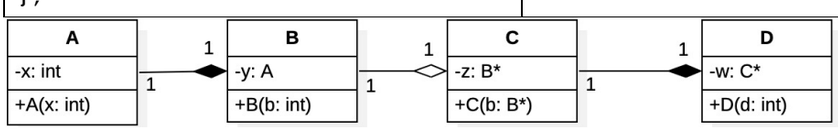
Consider partial definition of C++ classes A, B, C, D, and the following class diagram:

```
class A {
    int x;
public:
    A(int x): x(x){}
    int get_x()const {return x;}
};

class B {
    A y;
public:
    B(int b);
    A get_A()const {return y;}
};
```

```
class C {
    B* z;
public:
    C(B* b);
    const B* get_B()const {return z;}
};

class D {
    C* w;
public:
    D(int d);
};
```



Based on the given class diagram, write the proper implementation of constructors for classes: B, C, and D.

```
B::B(int b) : y(b) { }

C::C(B* b) : z(b) { }


D::D(int d) {
    w = new C(new B(d));
}
```

Question 3 – Inheritance

Consider the partial definition of class Quux, and class Bar that is derived from Quux.

```
// quux.h
class Quux {
protected:
    char *quux;
public:
    Quux(const char* q);
    ~Quux() {delete [] quux;}
    Quux(const Quux& src);
};
```

```
// bar.h
class Bar: public Quux {
protected:
    char *bar;
public:
    Bar(const char* b, const char *q);
    ~Bar() {delete [] bar;}
};
```

```
// quux.cpp

Quux::Quux(const char* q): quux(new
char[strlen(q)+1]) {
    if(quux == nullptr) {
        cout << "No memory available...";
        exit(1);
    }
    strcpy(this ->quux, q);
}
```

```
// bar.cpp
Bar::Bar(const char* b, const char* q):
Quux(q),bar(new char[strlen(b)+1]) {
    if(bar == nullptr) {
        cout << "No memory available...";
        exit(1);
    }
    strcpy(this ->bar, b);
}
```

Based on partial code given for both classes, write ONLY the implementation of copy constructor for class Bar.

```
Bar::Bar(const Bar& src) : Quux(src) {

    bar = new char[strlen(src.bar) + 1];

    if (bar == nullptr) {
        cout << "No memory available...";
        exit(1);
    }

    strcpy(this->bar, src.bar);
}
```

Question 4 – Aggregation and Composition in Java

As we learned during the lectures implementation of composition and aggregation is more or less similar to C++, but slightly different. Consider partial definition of Java classes A, B, C, D, and the following class diagram:

```
class A {
  int x;
  public A(int x){this.x = x;}
  public int get_x(){return x;}
}

class B {
  A y;
  public B(int b) {......}
  public A get_A(){return y;}
};
```

```
class C {
  B z;
  public C(B b) {......}
  public B get_B(){return z;}
}

class D {
  C w;
  public D(int d) { ......}
}
```

Based on relationships shown in the class diagram in question 2, write the proper implementation of constructors for classes: B, C, and D:

```
// Constructor for class B
public B(int b) {
    y = new A(b);  // Create a new A object with b
}
// Constructor for class C
public C(B b) {
    z = b;
}
// Constructor for class D
public D(int d) {
    w = new C(new B(d));
}
```

Question 5 – Non-public Derivation of Classes

Consider the partial definition of the following classes. You may assume classes A, B, and C are defined in the same file in the given order from left to right.

```
class A{
public:
    int a1;

protected:
    int a2;
private:
    int a3;
};
```

```
class C: private A{
public:
    int c1;
    void funC();
protected:
    int c2;
private:
    int c3;
};
```

```
class D: public C
{
public:
    int d1;
    void funD();
protected:
    int d2;
private:
    int d3;
};
```

In the following table if a data member is **NOT** accessible in member functions `funC` and `funD`, mark it with (**X**), otherwise leave it blank. Negative marks will be given to wrong answer. Negative marks will be considered for wrong answers.

	a1	a2	a3	c1	c2	c3	d1	d2	d3
funC			x				x	x	x
funD	x	x	x			x			

Question 6 : Consider the partial definition of class Text and the following C++ program. Then, answer the following question. You may assume all necessary header files are included and all member functions are properly implemented.

```
class Text{
    char* storageM;
    int lengthM;
public:
    Text(int n);
    /* REQUIES: n > 1;
     * PROMISES: allocates an array
     * of n characters on the heap and
     * sets '\0' to all of them.
     */
    Text(char* s);
    // Add function prototypes here



};
```

```
int main(){
  Text q = 20;
  q[1] = 'M';
  cout << "please enter word less than 20
           character: ";
  cin >> q;
  cout << q[0] << endl;
  return 0;
}
```

Write the definition of the overloaded operators that is needed for class Text, to allow the given main function work without any error. Marks will be deducted for writing UNECESSARY overloaded operators.

```
char& operator[](int index) const {  // must return char&
    return storageM[index];
}


istream& operator>>(istream& is, const Text& t) {
    is >> t.storageM;
    return is;
}
```

Question 7 - Multiple Inheritance:

Consider the partial implementation of the following C++ program, and the definition of classes A, B, C, D, and E.

```
class A{
protected: int a;
public: A(int a){this->a = a;}
};

class B: public A{
protected: int b;
public: B(int a, int b);
};

class C: virtual public B{
protected: int c;
public: C(int a, int c);
};

class D: virtual public B{
protected: int d;
public: D(int a, int d);
};

class E: public C, public D{
protected: int e;
public: E(int a, int b, int c);
};

int main(){
    E mye(1, 2, 3);
    return 0;
}
```

Write the implementation of constructor for class E. Please don't worry about the order of using arguments of this constructor. You don't have to write implementation of any other function.

```
E::E(int a, int b, int c): B(a, b), C(a, b), D(a, c), e(c) { }
```

Question 8 (4 marks): Consider the partial template definition of classes *Lis*t and *Node*, that maintains a list of objects of different types.

```
struct Node {
  T item;
  Node<T>*next;
};
```

```
template <class T>
class List {
public:
 List() {headM = nullptr;}
 void push_back(T& itemA);// appends a new node with itemA to the end of the list
 void reset() {cursor = headM;} // moves cursor to the beginning of the list
private:
 Node <T> *headM;
 Node <T> *cursor;
};
```

Assuming that we want to be able to use the list object as indicated here:

```
List <int> list;
list.push_back(10);
```

```
list.push_back(35);
list.push_back(60);
list.reset();
cout << ++list; // moves cursor to the second node and returns 35
cout << list++; // displays 35 and moves the
```

Based on the given code, above, write the implementation of the any overloaded operators which is needed. Marks will be deducted for writing UNECESSARY overloaded operators.

```
// Overload for prefix ++
List<T>& operator++() {
    if (cursor != nullptr && cursor->next != nullptr) {
        cursor = cursor->next;  // Move the cursor to the next node
    }
    return cursor->item; // returns next item
}

// Overload for postfix ++
T operator++(int) {
    T temp = cursor->item;  // Store the current item
    if (cursor != nullptr && cursor->next != nullptr) {
        cursor = cursor->next;  // Move the cursor to the next node
    }
    return temp;  // returns previous item
}
```

Question 9: Consider the following definition and implementation of class `TwoItem`. On the right side of the box **rewire** the `template` definition of the given code, in a way that both data members `first` and `second` will be of type **`T`**. Don't worry about specialization of any member function or the class itself.

```
class TwoItem{
  int first;
  int second;
public:
  TwoItem(int first, int second);
  int larger()const;
  friend ostream& operator <<
            (ostream&, const TwoItem&);
};

TwoItem::TwoItem(int first, int second){
  this->first = first;
  this->second = second;
}

inline int TwoItem::larger()const{
  return first > second?first:second;
}
// Don't worry about implementation of
// overloaded operator <<
```

```
// Template class definition
template <typename T>
class TwoItem {
  T first;
  T second;
public:
  TwoItem(T first, T second) {
     this->first = first;
     this->second = second;
  }

  T larger() const {
     return (first > second) ? first : second;
  }

};
```

Question 10: Consider the following Java program with the user defined class Vector:

```
class Vector implements Cloneable{
  int [] array;
  int size;
  public Vector(int size, String name) {
    array = new int[size];
    for(int i =0; i < size; i++ ) {
    array[i] = 100;
    }
  }
  public void set(int i, int value) {
    if(i >=0 || i < size)
    array[i] = value;
  }
  public String toString() {
    String temp = "Values in " + name + "are:";
    for(int item: array) {
    temp += item + " ";
    }
    return temp;
  }
  static public void main(String [] args) {
   Vector v1 = new Vector (3, "v1");
   System.out.println("Expected values in v1: 100 100 100");
   System.out.println("Actual values in v1: " + v1);
   Vector v2 = v1;
   System.out.println("Expected values in v2: 100 100 100");
   System.out.println("Actual values in v2: " + v2);
  v2.set(1, 300);
  System.out.println("Expected values in v1: 100 100 100");
  System.out.println("Actual values in v1: " + v1);
  }
}
```

If you run this program, it produces the following output:

```
Expected values in v1: 100 100 100
Actual values in v1: 100 100 100
Expected values in v2: 100 100 100
Actual values in v2: 100 100 100
Expected values in v1: 100 100 100
Actual values in v1: 100 300 100
```

Solution:

.

As you can see the expected value of v1 is different from its actual value. The low-level design of this code is defective. Explain what is wrong with this class and how can be the issue solved:

Without a clone method a shallow copy will be created. Solution is to implement cloneable and override the clone method.

```
@Override
protected Object clone() throws CloneNotSupportedException {
    Vector cloned = (Vector) super.clone();
    cloned.array = array.clone();
    return cloned;
}
```

Question 11: Consider the following C++ program is exactly like Java program in question 1.

```
class Vector {
  int* array;
  int size;
  public:
  Vector(int size) {
    array = new int[size];
    for(int i =0; i < size; i++ ) {
      array[i] = 100;
    }
    this->size = size;
  }

  void set(int i, int value) {
    if(i >=0 || i < size)
      array[i] = value;
  }

  string toString() {
    string temp = "";
    for(int i =0; i < size; i++) {
      temp += std::to_string(array[i]) + " " ;
    }
    return temp;
  }
};

int main(void) {
  Vector v1 (3);
  cout << "Expected values in v1: 100 100 100" << endl;
  cout << "Actual values in v1: " + v1.toString();
  Vector v2 = v1;
  cout << "Expected values in v2: 100 100 100" << endl;
  cout << "Actual values in v2: " + v2.toString();
  v2.set(1, 300);
  cout << "Expected values in v1: 100 100 100" << endl;
  cout << "Actual values in v1: " + v1.toString();
  return 0;
}
```

If you run this program, it produces the following output:

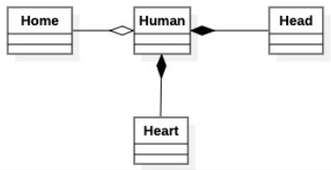
```
Expected values in v1: 100 100 100
Actual values in v1: 100 100 100
Expected values in v2: 100 100 100
Actual values in v2: 100 100 100
Expected values in v1: 100 100 100
Actual values in v1: 100 300 100
```

As you can see the expected value of v1 is different from its actual value. The low-level design of this code is defective. Explain what is wrong with this class and how can be the issue solved.

Similarly, we need to write a copy constructor.

```
Vector(const Vector& other) {
        size = other.size;
        array = new int[size];  // Allocate new memory
        for (int i = 0; i < size; i++) {
            array[i] = other.array[i];  // Copy each element
        }
}
```

Question 12: Consider the following class diagram and given implementation of classes: Human, Home, Head, Heart, and answer the following question.



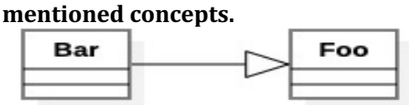
`class Head {`
`private:`
`  int i;`
`public:`
`  Head(int i);`
`};`
	`class Heart{`
	`private:`
	`  double d;`
	`public:`
	`  Heart(int d);`
	`};`
		`class Home {`
		`private:`
		`  char c;`
		`public:`
		`  Home(char c);`
		`};`
			`class Human {`
			`private:`
			`  Head head;`
			`  Heart* heart;`
			`  Home* home;`
			`public:`
			`};`

`Head::Head(int i){`
`  this->i = i;`
`}`
	`Heart::Heart(double d){`
	`  this->d = d;`
	`}`
		`Home::Home(char c){`
		`  this->c = c;`
		`}`
			`int main(){`
			` Home h('C');`
			` Human human(100, 555.5, &h);`
			`}`

Write the implementation of constructor for class Human:

```
Human(int i, double d, Home& h) : head(i){
   heart = new Hear(d);
   home = &h;
}
```

Question 13 (5 marks): Consider The following class diagram and C++ program and answer the following question. You can assume all required header file are included. This section is concerned about refactoring/changing the given code to allow the concepts such as inheritance and polymorphism work, properly. **Note: Marks will be deducted if you suggest any unnecessary changes that will not affect the above-mentioned concepts.**



	// file foo.h		// file foo.cpp
1	class Foo{	10	void Foo::fun1() {
2	private:	11	cout << *xM << endl;
3	int* xM;	12	}
4	public:	13	
5	Foo(int n);	14	Foo::Foo(int n) {
6	~Foo ();	15	xM = new int [n];
7	void fun1();	16	}
8	void fun2();	17	
9	};	18	Foo::~Foo() { delete xM;}
			void Foo::fun2() {
			(*xM)++;
			}
	// file bar.h		// file bar.cpp
19	class Bar {	26	Bar::Bar(int n, int m): yM(new int [m]){
20	protected:	27	}
21	int *yM;	28	
22	public:	29	Bar::~Bar() {
23	Bar(int n, int m);	30	delete yM;
24	~Bar();	31	}
25	void fun1();	32	
25	};	33	void Bar::fun1() {
		34	cout << *xM << " " << *yM << endl;
			}

Part a – Does any line from 1 to 3 needs any changes to improve the code for inheritance or polymorphism? If yes, explain briefly but clearly:

```
Line 2 needs to be protected
```

Part b – Does any line from 4 to 8 needs any changes to improve the code for inheritance or polymorphism? If yes, explain briefly but clearly:

```
Line 6 needs to be:  virtual ~Foo()
Line 7 needs to be: virtual fun1();
```

Part c – Does any line from 19 to 25 needs any changes to improve the code for inheritance or polymorphism? If yes, explain briefly but clearly:

```
Line 19: should change to class Bar: public Foo { ... }
```

Part d – Does any line from 26 to 34 needs any changes to improve the code for inheritance or polymorphism? If yes, explain briefly but clearly:

```
Bar::Bar(int n, int m) : Foo(n), yM(new int[m]) {
}
```

Multiple Choice Questions

Please write your answer on the scantron answer sheet. Answers written on the exam paper will not be accepted:

Consider the following C++ program. Assuming constructor, copy constructor, assignment operator, and destructor of this class are properly implemented, answer the following questions:

```
class Box {
public:
    Box();
    ~Box();
    Box(const Box& source);
    Box& operator=(const Box& rhs);
private:
    int* pointer;
};
```

```
int main(void){
    Box x;
    Box y(x);
    Box *z = new Box;
    x = y;
    return 0;
}
```

1. **How many times the constructor of class Box is called?**
   a) Once
   b) Twice
   c) Three times
   d) None of the above

2. How many times the copy constructor of class Box is called?
   a) Once
   b) Twice
   c) Three times
   d) None of the above

3. How many times destructor of class Box is called?
   a) Once
   b) Twice
   c) Three times
   d) None of the above

Consider the partial definition of the following C++ classes, and the given main function, to answer the following question.

```
class A{
  char * s1;
public:
  A(int n){
    s1= new char[n];
  }
  ~A() { delete [] s1;}
};
```

```
class B : public A {
  char * s2;
public:
  B(int n):A(n) {
     s2=new char[n];
  }
  ~B() { delete [] s2;}
};
```

```
1  int main(void){
2    A *p1 = new B(5);
3    B *p2 = new B(6);
4    p1 = p2;
5    p2 = p1;
6    // MORE CODE HERE
7    return 0;
8  }
```

4. The above C++ program has a **compilation error** in the main function, in:
   a. Line 2
   b. Line 3
   c. Line 4
   d. Line 5
   e. Line 6

Consider the partial definition of the following C++ classes, and the given `main` function, to answer the following question.

```
class A{
  char * s1;
public:
  A(int n){
    s1= new char[n];
  }
  ~A() { delete [] s1;}
};
```

```
class B : public A {
  char * s2;
public:
  B(int n):A(n) {
     s2=new char[n];
  }
  ~B() { delete [] s2;}
};
```

```
1  int main(void){
2    A *p = new B(5);
3    // MORE CODE HERE
4    Delete p;
5    return 0;
6  }
7
8
```

5. There is a serious memory management issue in this program. Which one of the followings addresses the issue:
   a. Constructor of class B is not properly defined
   b. `p is` not properly deleted.
   c. Destructor of class A is not declared virtual
   d. None of the above

6. Consider the definition of the class String in C++ and assume all member functions are properly defined:

```
class String {
```

```
    public:
        String(int a = 0);
        String(const char* b);
    private:
        int length;
        char* storage;
};
```

Which of the following statements is a **valid** statement?

a. `String s1(50);`
b. `String s2 = 100;`
c. `String s3;`
d. `String s4 = "123";`
e. `All of the above are valid statements`

7. Which one of the following statements is a correct statement in C++?
    a. An overloaded operator function must have at least a class object as its argument.
    b. An overloaded operator cannot return a pointe, but can return a reference.
    c. All of the above are correct answers
    d. None of the above is a correct answer
8. Class A declares class B as friend. In the above program:
    A. Class A can have access to all data members in class B.
    B. Class B can have access to all data members in class A.
    C. Class B can have access to all data members of in class A and class A can have access to data member in class B.
    D. None of the above is correct.
9. Assume in a C++ class called Text we want to overload many operators. Which one of the following group of operators must be overloaded as a non-member function?
    a. `&&, ||, new`
    b. `++, and --`
    c. `<<, >>`
    d. `All of the above`
    e. `None of the above.`
10. Strong coupling leads to a program that:
    a. Is easier to maintain.
    b. Is more efficient.
    c. Is not easier to maintain but it is more efficient.
    d. None of the above
11. Object-oriented software development supports two types of hierarchy which includes:
    a. Class hierarchy and entity hierarchy
    b. Whole-part hierarchy and Generalization-Specialization hierarchy
    c. Class hierarchy and structure chart
    d. Class hierarchy and process hierarchy
    e. None of the above

Consider the following problem statement**:**
*A Hockey-League is made up of at least four Hockey Teams. Each Hockey-Team has six to twelve players, and one of the players assumes the **role** of a Captain (Note: Captain has no additional attribute and no additional functionality – his/her attributes and functionalities are identical to other players). Hockey teams play games against each other. Teams are sometimes led by a coach, and he/she can coach multiple teams.*

Assuming we want to design Java or C++ classes such as Game, Team, League, Player, Coach, for the give problem statement, answer the following questions:

12. Which one of the following relationships is **the best** conceptual representation of the relationship between Team and Player:
    a. Aggregation, with no multiplicity
    b. Aggregation, with one-to-many multiplicity
    c. Inheritance
    d. Association
    e. None of the above
13. Which one of the following relationships is **the best** conceptual representation of the relationship between Team and Game:
    a. Composition, with one-to-many multiplicity
    b. Inheritance
    c. Association, with no multiplicity
    d. Association, with one-to-many Association
14. Which one of the following relationships is **the best** conceptual representation of the relationship between Player and Captain:
    a. Composition, with one-to-many multiplicity
    b. Inheritance
    c. Association, with no multiplicity
    d. Association, with one-to-many Association
    e. None of the above

15. A static data member in a C++ class:
    a. Must be declared globally outside the class definition
    b. Will be initialized to zero automatically.
    c. All of the above: A, B, and C are all correct
    d. None of the above are correct.

16. Which one of the following statements is correct? Select the best answer.
    a. An abstract class is a class that contains a virtual function.
    b. An abstract class is a class that contains at least one pure virtual function.
    c. An abstract class is class that is derived from a virtual base class.
    d. A, and B are correct answers.
    e. All of the above are correct answers.
    f. None of the above is a correct answer.

17. Which one of the following statements is correct about a static member function in a C++ class? Select the best answer.
    a. A static member function can be called even without existence of a class object.
    b. A static member function doesn't have direct access to the class private data members.
    c. A static member function does not have a 'this pointer.'
    d. B and C are correct answers
    e. All the above are correct answers.
    f. None of the above

18. Which one of the following statements are correct?
    a. An abstract class is a class that is at the root of the entire class hierarchy
    b. An abstract class is a class that cannot have an object.
    c. An abstract class can be a reference or pointer argument of a global function
    d. B and C are both correct.
    e. All the above are correct.

19. Which one of the following statements is a correct in C++?
    a. An overloaded operator function must be always a member of a class or a friend of a class
    b. An overloaded operator function must have at least a class objec as its argument.
    c. An overloaded operator cannot return a pointer, but can return a reference.
    d. A and B are correct answers.

Consider the following PaymentStrategy interface and two concrete classes CreditCardPayment, PayPalPayment, and ShoppingCart:

```
// Strategy Interface
interface PaymentStrategy {
    void pay(int amount);
}


// Concrete Strategy 1: Credit Card Payment
class CreditCardPayment implements PaymentStrategy {
    @Override
    public void pay(int amount) {
        System.out.println("Paid " + amount + " using Credit Card.");
    }
}

// Concrete Strategy 2: PayPal Payment
class PayPalPayment implements PaymentStrategy {
    @Override
    public void pay(int amount) {
        System.out.println("Paid " + amount + " using PayPal.");
    }
}

class ShoppingCart {
    private PaymentStrategy paymentStrategy;

    public void setPaymentStrategy(PaymentStrategy paymentStrategy) {
        this.paymentStrategy = paymentStrategy;
    }

    public void checkout(int amount) {
        paymentStrategy.pay(amount);
    }
}
```

Design Pattern is not part of midterm

20. Which of the following statements about the code below is **incorrect**?

```
ShoppingCart cart = new ShoppingCart();
cart.setPaymentStrategy(new PayPalPayment());
cart.checkout(100);
```

    a. The code sets PayPalPayment as the payment method at runtime.
    b. The checkout method will call the pay method of the PayPalPayment class.

c. The checkout method can only work with the PayPalPayment class.

d. The payment method can be changed to another strategy (like CreditCardPayment) without modifying the ShoppingCart class

Design Pattern is not part of midterm

21. Consider the definition of class Vector. Which one of the following is the correct specialisation of its constructor:

```
template <class T>
class Vector
{
public:
  Vector(int s);
  ~Vector();
  T getValue(int elem);
  void display();
private:
  T *array;
  int size;
};
// Class specialisation:
template <>
class Vector<char*> {
  int size;
  char** ar;
  public:
  Vector(int s);
  void display();
};
```

a.

```
Vector <char* >::Vector(int s) {
  size = s;
  array = new char*[size];
  assert (array != nullptr);
  strcpy(array, s)
}
```

b.

```
template <char*>
Vector::Vector(int s) {
  size = s;
  array = new char*[size];
  assert (array != nullptr);
  strcpy(array, s)
}
```

c.

```
template <>
Vector <char* >::Vector(int s) {
  size = s;
  array = new char*[size];
  assert (array != nullptr);
  strcpy(array, s)
}
```

d.

```
Vector ::Vector<char*>(int s) {
  size = s;
  array = new char*[size];
  assert (array != nullptr);
  strcpy(array, s)
}
```

22. Which one of the following statements is correct?

a. An overloaded operator cannot be declared static.

b. Operators << and >> cannot be defined as a member of a class.

c. All of the above are correct answers.

d. None of the above are correct answers.

23. Consider the definition of the class String in C++ and assume all member functions are properly defined:

```
class String {
   public:
      String(int a = 0);
       String(const char* b);
   private:
       int length;
       char* storage;
       };
```

Which of the following statements is a **valid** statement?

A. `String s1(50);`
B. `String s2 = 100;`
C. `String s3;`
D. A and C are the only valid statements
E. A, B, and C are all valid statements.

24. Which one of the following group of operators must be overloaded as a member function in a C++ class?
    a. `==, >=, <=`
    b. `&&, ||, new`
    c. `++, and --`
    d. `(), =, []`
    e. `All of the above`