1. (6.9) Multidimensional arrays can be stored in row major order, as in C++, or in column major order, as in Fortran. Develop the access functions for both of these arrangements for three-dimensional arrays.
   a = array[x][y][z]; data size = d;

   Row major:
   a[l][m][n] = address of a + (l + x(m + y(n + z))) * d

   Column major:
   a[l][m][n] = address of a + (n + z(m + y(l + x))) * d

2. (11.2) Suppose someone designed a stack abstract data type in which the function top returned an access path (or pointer) rather than returning a copy of the top element. This is not a true data abstraction. Why? Give an example that illustrates the problem.

這違反了資料抽象的關鍵原則，即資訊隱藏和封裝。

資料抽象的目的是將資料結構的內部實作細節對外部使用者隱藏起來，使用者只能透過明確定義的抽象操作（如 push、pop、top）來與資料結構互動。如果 top 回傳一個指標，它就洩露了堆疊內部記憶體管理或表示方式的細節。這使得外部使用者能夠：

繞過抽象操作：使用者可以直接透過這個指標修改堆疊頂端元素的值，而不需要呼叫任何堆疊的抽象操作。

破壞資料結構的完整性：使用者可能使用這個指標進行不當操作，例如釋放記憶體、存取陣列越界等，從而破壞堆疊內部的一致性和狀態。

假設堆疊是使用一個陣列和一個**stack pointer**實作的。

操作	真正的抽象資料型別 (top 回傳副本)	違背抽象資料型別的設計 (top 回傳指標)
堆疊初始化	Stack S = [10, 20, 30](堆疊頂端是 30)	Stack S = [10, 20, 30] (堆疊頂端是 30)
取得堆疊頂端	int val = S.top(); ->val = 30	int* ptr = S.top(); ->ptr 指向儲存 30 的記憶體地址
修改	val = 99;	*ptr = 99;
堆疊狀態	S 仍為[10, 20, 30]。 堆疊的內部狀態未被改變。	S 變為[10, 20, 99]。堆疊頂端元素在不呼叫 push 或任何修改操作的情況下被修改了。

在這個例子中，透過回傳的指標 ptr，使用者能夠直接修改堆疊的內部資料（將 30 改為 99），而沒有經過堆疊的抽象介面。這使得堆疊抽象層無法控制或追蹤其內部狀態的變化，從而破壞了抽象的完整性。

3. (12.6) Compare the multiple inheritance of C++ with that provided by interfaces in Java.
   C++ 的多重繼承 (Multiple Inheritance) 允許一個類別繼承多個父類別的實作 (Implementation) 和狀態 (State)；而 Java 提供的介面 (Interfaces) 則允許一個類別實作 (Implement) 多個介面的行為合約 (Behavioral Contract)。這兩種機制的主要區別在於：C++ 允許繼承「是什麼」(is-a 關係和實作細節)，而 Java 介面只允許繼承「能做什麼」(can-do 關係和類型)。

4. (12.16) What is the primary reason why all Java objects have a common ancestor?

所有 Java 物件都擁有一個共同祖先（即 java.lang.Object 類別）的主要原因，是為了提供一套普遍、標準化的基礎行為與能力，這是所有在 Java 虛擬機器（JVM）中運行的物件都必須具備的。

這項設計確保了以下幾點：

A. 普遍合約與多型性 (The Core Benefit)
B. 核心系統服務的實現
C. 程式碼重複使用與慣例

5. (12.19) What are the differences between a C++ abstract class and a Java interface?

狀態（實例變數）的有無

- C++ 抽象類別：可以包含非靜態資料成員（即狀態）。這使得抽象類別能夠儲存所有子類別共享的屬性，並且其具體方法可以操作這些狀態。
- Java 介面：不能包含實例變數。它只能包含常數（public static final 欄位）。因此，介面的任何 default 方法都不能直接存取實作類別的實例狀態，而必須透過呼叫其他抽象或具體方法間接獲取狀態。

繼承機制 (多重性)

- Java：採用單一繼承（類別只能 extends 一個父類別或抽象類別），因此必須使用多個介面（implements）來獲得多重行為合約，以避免 C++ 多重繼承帶來的「菱形問題」。
- C++：由於語言本身支援多重繼承，一個類別可以直接從多個基底類別（無論是否抽象）繼承實作和狀態。

建構子的作用

- C++ 抽象類別：可以有建構子。儘管抽象類別不能被實例化，但這些建構子會被子類別呼叫，用於初始化抽象類別中定義的資料成員（狀態）。
- Java 介面：沒有建構子的概念，因為它們沒有實例狀態需要初始化。

具體實作 (Implementation)

- C++ 抽象類別：可以包含大量已實作的非虛擬和虛擬方法，為子類別提供一個骨架實作 (Skeletal Implementation)。
- Java 介面：傳統上是純抽象的。雖然現在有了 default 方法，但它們的主要目的是擴展合約，且無法存取子類別的實例狀態。因此，它們通常只用於提供簡單的便利方法或預設行為。

6. (12.24) Study and explain the issue of why Java does not include C++'s destructor even though both these languages have constructors.
   Java 透過 GC(Garbage Collection) 消除了手動記憶體管理的負擔，並透過 try-with-resources 機制實現了對非記憶體資源的可靠、自動化清理，從而無需 C++ 的 destructor。