OOP HW2

Notes

2026 年 5 月 8 日

1 PTA2

Question 1

The copy constructors can be used to _____

- A. Copy an object so that it can be passed to another primitive type variable
- B. Copy an object for type casting
- **C. Copy an object so that it can be passed to a function**
- D. Copy an object so that it can be passed to a class

解析：正确选项是 **C**。当对象以“值传递（Pass by Value）”的方式作为形参传入函数，或者从函数中按值返回时，编译器会自动调用拷贝构造函数（Copy Constructor）来生成该对象的一个克隆副本，以避免原始对象的数据被函数内部意外破坏。

Question 2

Which constructor will be called from the object obj2 in the following C++ program?

```
A obj2(10,20);
```

- **A. A(int y, int x)**
- B. A(int y; int x)
- C. A(int y)
- D. A(int x)

解析：正确选项是 **A**。对象实例化时，编译器会根据传入参数的数量、类型和顺序进行函数重载决议（Overload Resolution）。`obj2` 在创建时明确传入了两个整型参数 `(10, 20)`，因此精确匹配带有两个 `int` 形参的构造函数 `A(int y, int x)`。

Question 5

Which of the following is not a property of an object?

- A. Properties
- **B. Names**
- C. Identity
- D. Attributes

解析：正确选项是 **B**。在面向对象编程（OOP）理论中，对象包含三个核心要素：状态（State / Properties / Attributes）、行为（Behavior / Methods）和标识（Identity，例如内存地址）。“名字（Names）”仅仅是代码中程序员用于引用该对象的变量标识符，它不是对象本身固有的、独立于程序运行的内在属性。

Question 10

If data members are private, what can we do to access them from the class object?

- A. Private data members can never be accessed from outside the class
- **B. Create public member functions to access those data members**
- C. Create private member functions to access those data members
- D. Create protected member functions to access those data members

解析：正确选项是 **B**。这是面向对象“封装（Encapsulation）”特性的标准实践：将数据成员设为 `private` 隐藏在类内部，同时暴露 `public` 的成员函数（如常见的 Getter 和 Setter 方法）作为合法接口，供外部代码安全地读取和修改这些数据。

Question 15

Copy constructor will be called whenever the compiler _____

- A. Generates implicit code
- B. Generates member function calls
- **C. Generates temporary object**
- D. Generates object operations

解析：正确选项是 **C**。C++ 编译器在底层处理对象的按值传参、按值返回，或者执行某些隐式类型转换时，通常会在内存的栈区生成临时对象（Temporary object）。每当这个临时对象被创建并需要用另一个现存对象来初始化时，就必定会触发拷贝构造函数的调用。

Question 17

Can a copy constructor be made private?

- **A. Yes, always**
- B. Yes, if no other constructor is defined
- C. No, never
- D. No, private members can't be accessed

解析：正确选项是 **A**。在 C++ 中（尤其是 C++11 标准之前），将拷贝构造函数和赋值运算符显式声明为 `private` 且不提供实现，是一种极其经典的防止对象被错误复制的模式（即打造 Non-copyable Object，如互斥锁、单例对象等）。如今虽更推荐使用 `= delete`，但将其设为私有在语法上永远是合法的。

Question 20

Does constructor overloading include different return types for constructors to be overloaded?

- A. Yes, if return types are different, signature becomes different
- B. Yes, because return types can differentiate two functions
- C. No, return type can't differentiate two functions
- **D. No, constructors doesn't have any return type**

解析：正确选项是 **D**。构造函数是一种极其特殊的成员函数，它在语法上根本不允许定义任何返回类型（连 `void` 都不允许写）。因此，既然没有返回类型，自然也就谈不上“依靠返回类型的不同来进行重载”。构造函数的重载只能通过参数列表（形参类型、个数、顺序）的不同来实现。

Question 21

Destructor calls _____ (C++)

- A. Are only implicit
- B. Are only explicit
- **C. Can be implicit or explicit**
- D. Are made at end of program only

解析：正确选项是 **C**。析构函数通常在对象离开其作用域或生命周期结束时，由系统**隐式调用（Implicit）**。但在某些高级内存管理场景下（例如使用了定位 `new` / Placement `new` 在预分配的内存上构造对象时），程序员必须通过 `obj.~ClassName()` 的语法来**显式调用（Explicit）**析构函数以清理资源。

Question 26

When a destructor is called?

- A. After the end of object life
- B. Anytime in between object’s lifespan
- C. At end of whole program
- **D. Just before the end of object life**

解析：正确选项是 **D**。析构函数的调用时机非常精确：它是在对象的生命周期即将终结，但其所占用的内存空间还未被系统彻底回收的“最后一刻（Just before）”被执行。它的核心任务是在对象彻底灰飞烟灭前，安全地释放掉它生前持有的外部资源（如堆内存、文件句柄等）。

Question 29

Global destructors execute in _____ order after main function is terminated.

- A. Sequential
- B. Random
- **C. Reverse**
- D. Depending on priority

解析：正确选项是 **C**。C++ 中对象的构造和析构遵循栈（Stack）的“后进先出（LIFO）”严格逻辑。因此，无论是在局部函数中还是在全局存储区，对象的析构顺序总是与其在初始化时构造的顺序**完全相反（Reverse order）**。

Question 31

Which among the following is true for public class?

- A. There can be more than one public class in a single program
- B. Public class members can be used without using instance of class
- C. Public class is available only within the package
- **D. Public classes can be accessed from any other class using instance**

解析：正确选项是 **D**。（注：这道题干带有强烈的 Java/C# 面向对象风格）。公开类（Public Class）的设计初衷就是向整个程序的其他模块暴露其定义。在任何其他类或作用域中，只要你创建了该公开类的实例（Instance/对象），就可以自由地访问其公共属性和方法。

Question 38

What is the output of following code?

```
int n=10; // global
class A {
private :
    int n;
public :
    int m;
    A() { n=100; m=50; }
    void disp() { cout<<"n"<<m<<n; }
};
```

- A. 1050100
- B. 1005010
- C. n5010
- **D. n50100**

解析：正确选项是 **D**。本题考查名字查找和局部优先原则。在 `disp()` 的输出流中：第一个 `"n"` 是带有双引号的字符串字面量，原样输出字符 `n`；接着输出 `m` 的值 `50`；最后输出变量 `n`。根据 C++ 的**局部优先原则（Name Shadowing/Hiding）**，类内部的成员变量 `n=100` 的作用域覆盖了全局变量 `n=10`。因此最终拼接的输出结果为 `n50100`。

Question 44

Can a function, other than the enclosing function of local class, access the class members?

- A. Yes, using object
- B. Yes, using direct call
- C. Yes, using pointer
- **D. No, can’ t access**

解析：正确选项是 **D**。局部类（Local Class）是定义在某一个特定函数体内部的类。它的作用域被死死限制在包裹它的那个函数的花括号内，对外部世界完全隔离（作用类似于局部变量）。因此，定义它的外部函数之外的其他任何函数，甚至连它的名字都无法识别，更别提访问其成员了。

Question 45

Which among the following is the main advantage of using local classes?

- A. Make program more efficient
- B. Makes program execution faster
- **C. Helps to add extra functionality to a function**
- D. Helps to add more members to a function

解析：正确选项是 **C**。局部类并不能提升程序的运行速度或效率。它的核心价值在于代码结构的管理：当你编写一个复杂的函数时，可以临时在内部定义一个小型的辅助数据结构，用来提供该函数专属的额外功能。这能完美避免污染全局的命名空间，极大地提高了代码的内聚性和可读性。

Question 48

Non-static nested classes have access to _____ from enclosing class.

- A. Private members
- B. Protected members
- C. Public members
- **D. All the members**

解析：正确选项是 **D**。在 C++ 访问控制规则中，嵌套类（Nested Class）被编译器完全视同为外部类的一个内部成员。由于“成员可以访问本类的所有其他成员”，因此嵌套类享有极高的权限，它可以无条件访问包围它的外部类的**所有成员（包括 private 和 protected 成员）**。

Question 56

Which among the following is correct?

- **A. Friend function of derived class can access non-private members of base class**
- B. Friend function of base class can access derived class members
- C. Friend function of derived class can access members of only derived class
- D. Friend function can access private members of base class of a derived class

解析：正确选项是 **A**。友元关系是不具备继承性的。子类（派生类）的友元函数可以访问子类的所有成员；而父类（基类）的非私有成员（`public` 和 `protected`）已经被子类成功继承下来，成为了子类自身组成部分。因此，子类的友元函数自然能够顺理成章地访问这些通过继承得来的父类非私有成员。

Question 1

The copy constructors can be used to _____

- A. Copy an object so that it can be passed to another primitive type variable
- B. Copy an object for type casting
- **C. Copy an object so that it can be passed to a function**
- D. Copy an object so that it can be passed to a class

解析：正确选项是 **C**。当对象以“值传递（Pass by Value）”的方式作为形参传入函数，或者从函数中按值返回时，编译器会自动调用拷贝构造函数（Copy Constructor）来生成该对象的一个克隆副本，以避免原始对象的数据被函数内部意外破坏。

Question 2

Which constructor will be called from the object obj2 in the following C++ program?

```
A obj2(10,20);
```

- **A. A(int y, int x)**
- B. A(int y; int x)
- C. A(int y)
- D. A(int x)

解析：正确选项是 **A**。对象实例化时，编译器会根据传入参数的数量、类型和顺序进行函数重载决议（Overload Resolution）。`obj2` 在创建时明确传入了两个整型参数 `(10, 20)`，因此精确匹配带有两个 `int` 形参的构造函数 `A(int y, int x)`。

Question 5

Which of the following is not a property of an object?

- A. Properties
- **B. Names**
- C. Identity
- D. Attributes

解析：正确选项是 **B**。在面向对象编程（OOP）理论中，对象包含三个核心要素：状态（State / Properties / Attributes）、行为（Behavior / Methods）和标识（Identity，例如内存地址）。“名字（Names）”仅仅是代码中程序员用于引用该对象的变量标识符，它不是对象本身固有的、独立于程序运行的内在属性。

Question 10

If data members are private, what can we do to access them from the class object?

- A. Private data members can never be accessed from outside the class
- **B. Create public member functions to access those data members**
- C. Create private member functions to access those data members
- D. Create protected member functions to access those data members

解析：正确选项是 **B**。这是面向对象“封装（Encapsulation）”特性的标准实践：将数据成员设为 `private` 隐藏在类内部，同时暴露 `public` 的成员函数（如常见的 Getter 和 Setter 方法）作为合法接口，供外部代码安全地读取和修改这些数据。

Question 15

Copy constructor will be called whenever the compiler _____

- A. Generates implicit code
- B. Generates member function calls
- **C. Generates temporary object**
- D. Generates object operations

解析：正确选项是 **C**。C++ 编译器在底层处理对象的按值传参、按值返回，或者执行某些隐式类型转换时，通常会在内存的栈区生成临时对象（Temporary object）。每当这个临时对象被创建并需要用另一个现存对象来初始化时，就必定会触发拷贝构造函数的调用。

Question 17

Can a copy constructor be made private?

- **A. Yes, always**
- B. Yes, if no other constructor is defined
- C. No, never
- D. No, private members can’t be accessed

解析：正确选项是 **A**。在 C++ 中（尤其是 C++11 标准之前），将拷贝构造函数和赋值运算符显式声明为 `private` 且不提供实现，是一种极其经典的防止对象被错误复制的模式（即打造 Non-copyable Object，如互斥锁、单例对象等）。如今虽更推荐使用 `= delete`，但将其设为私有在语法上永远是合法的。

Question 20

Does constructor overloading include different return types for constructors to be overloaded?

- A. Yes, if return types are different, signature becomes different
- B. Yes, because return types can differentiate two functions
- C. No, return type can't differentiate two functions
- **D. No, constructors doesn't have any return type**

解析：正确选项是 **D**。构造函数是一种极其特殊的成员函数，它在语法上根本不允许定义任何返回类型（连 `void` 都不允许写）。因此，既然没有返回类型，自然也就谈不上“依靠返回类型的不同来进行重载”。构造函数的重载只能通过参数列表（形参类型、个数、顺序）的不同来实现。

Question 21

Destructor calls _____ (C++)

- A. Are only implicit
- B. Are only explicit
- **C. Can be implicit or explicit**
- D. Are made at end of program only

解析：正确选项是 **C**。析构函数通常在对象离开其作用域或生命周期结束时，由系统**隐式调用（Implicit）**。但在某些高级内存管理场景下（例如使用了定位 `new` / Placement `new` 在预分配的内存上构造对象时），程序员必须通过 `obj.~ClassName()` 的语法来**显式调用（Explicit）**析构函数以清理资源。

Question 26

When a destructor is called?

- A. After the end of object life
- B. Anytime in between object's lifespan
- C. At end of whole program
- **D. Just before the end of object life**

解析：正确选项是 **D**。析构函数的调用时机非常精确：它是在对象的生命周期即将终结，但其所占用的内存空间还未被系统彻底回收的“最后一刻（Just before）”被执行。它的核心任务是在对象彻底灰飞烟灭前，安全地释放掉它生前持有的外部资源（如堆内存、文件句柄等）。

Question 29

Global destructors execute in _____ order after main function is terminated.

- A. Sequential
- B. Random
- **C. Reverse**
- D. Depending on priority

解析：正确选项是 **C**。C++ 中对象的构造和析构遵循栈（Stack）的“后进先出（LIFO）”严格逻辑。因此，无论是在局部函数中还是在全局存储区，对象的析构顺序总是与其在初始化时构造的顺序**完全相反（Reverse order）**。

Question 31

Which among the following is true for public class?

- A. There can be more than one public class in a single program
- B. Public class members can be used without using instance of class
- C. Public class is available only within the package
- **D. Public classes can be accessed from any other class using instance**

解析：正确选项是 **D**。（注：这道题干带有强烈的 Java/C# 面向对象风格）。公开类（Public Class）的设计初衷就是向整个程序的其他模块暴露其定义。在任何其他类或作用域中，只要你创建了该公开类的实例（Instance/对象），就可以自由地访问其公共属性和方法。

Question 38

What is the output of following code?

```
int n=10; // global
class A {
private :
    int n;
public :
    int m;
    A() { n=100; m=50; }
    void disp() { cout<<"n"<<m<<n; }
};
```

- A. 1050100

- B. 1005010
- C. n5010
- **D. n50100**

解析：正确选项是 **D**。本题考查名字查找和局部优先原则。在 `disp()` 的输出流中：第一个 `"n"` 是带有双引号的字符串字面量，原样输出字符 `n`；接着输出 `m` 的值 `50`；最后输出变量 `n`。根据 C++ 的**局部优先原则（Name Shadowing/Hiding）**，类内部的成员变量 `n=100` 的作用域覆盖了全局变量 `n=10`。因此最终拼接的输出结果为 `n50100`。

Question 44

Can a function, other than the enclosing function of local class, access the class members?

- A. Yes, using object
- B. Yes, using direct call
- C. Yes, using pointer
- **D. No, can't access**

解析：正确选项是 **D**。局部类（Local Class）是定义在某一个特定函数体内部的类。它的作用域被死死限制在包裹它的那个函数的花括号内，对外部世界完全隔离（作用类似于局部变量）。因此，定义它的外部函数之外的其他任何函数，甚至连它的名字都无法识别，更别提访问其成员了。

Question 45

Which among the following is the main advantage of using local classes?

- A. Make program more efficient
- B. Makes program execution faster
- **C. Helps to add extra functionality to a function**
- D. Helps to add more members to a function

解析：正确选项是 **C**。局部类并不能提升程序的运行速度或效率。它的核心价值在于代码结构的管理：当你编写一个复杂的函数时，可以临时在内部定义一个小型的辅助数据结构，用来提供该函数专属的额外功能。这能完美避免污染全局的命名空间，极大地提高了代码的内聚性和可读性。

Question 48

Non-static nested classes have access to _____ from enclosing class.

- A. Private members
- B. Protected members
- C. Public members
- **D. All the members**

解析：正确选项是 **D**。在 C++ 访问控制规则中，嵌套类（Nested Class）被编译器完全视同为外部类的一个内部成员。由于“成员可以访问本类的所有其他成员”，因此嵌套类享有极高的权限，它可以无条件访问包围它的外部类的**所有成员（包括 private 和 protected 成员）**。

Question 56

Which among the following is correct?

- **A. Friend function of derived class can access non-private members of base class**
- B. Friend function of base class can access derived class members
- C. Friend function of derived class can access members of only derived class
- D. Friend function can access private members of base class of a derived class

解析：正确选项是 **A**。友元关系是不具备继承性的。子类（派生类）的友元函数可以访问子类的所有成员；而父类（基类）的非私有成员（`public` 和 `protected`）已经被子类成功继承下来，成为了子类自身组成部分。因此，子类的友元函数自然能够顺理成章地访问这些通过继承得来的父类非私有成员。

Question 61

What are the constant member functions?

- **A. Functions which doesn’t change value of calling object**
- B. Functions which doesn’t change value of any object inside definition
- C. Functions which doesn’t allow modification of any object of class
- D. Functions which doesn’t allow modification of argument objects

解析：正确选项是 **A**。常成员函数（即在函数声明末尾加上 `const` 关键字的函数，例如 `void func() const;`）是在向编译器郑重承诺：该函数内部绝不会修改调用该对象（即隐含的 `this` 指针所指向的对象）的任何数据成员状态。它是 C++ 保障数据安全的重要手段。

Question 65

Which type of member functions get inherited in the same specifier in which the inheritance is done?

- A. Private member functions
- B. Protected member functions
- **C. Public member functions**
- D. All member functions

解析：正确选项是 **C**。这考查的是 C++ 继承权限修饰符的核心规律：无论哪种继承方式，父类的 `private` 成员在子类中永远不可见。而父类的 `public` 成员在继承后，其在子类中的最终权限完全由继承方式（specifier）决定——`public` 继承依然是 `public`，`protected` 继承变成 `protected`，`private` 继承变成 `private`。

Question 73

If static data members have to be used inside a class, those member functions _____ (when no object exists)

- A. Must not be static member functions
- B. Must not be member functions
- **C. Must be static member functions**
- D. Must not be member function of corresponding class

解析：正确选项是 **C**。静态数据成员的生命周期独立于任何对象，程序一开始就存在。如果在尚未实例化任何对象的情况下，想要在类内部操作这些静态数据，就必须借助静态成员函数。因为普通成员函数必须依赖对象（需要隐含的 `this` 指针）才能被调用。

Question 78

If object of class are created, then the static data members can be accessed _____

- A. Using dot operator
- B. Using arrow operator
- C. Using colon
- **D. Using dot or arrow operator**

解析：正确选项是 **D**。虽然静态成员属于整个类，业界最规范的推荐写法是使用作用域解析符 `ClassName::` 访问。但在 C++ 语法设计中，为了方便和兼容，如果你已经持有了该类的对象实例或对象指针，同样完全允许使用普通的点（`.`）操作符或指针的箭头（`->`）操作符去访问静态成员。

Question 80

Which among the following is wrong syntax related to static data members?

- A. className :: staticDataMember;
- B. dataType className :: memberName =value;
- C. static dataType memberName;
- **D. className : dataType -> memberName;**

解析：正确选项是 **D**。这是一个明显的“生造”干扰项。合法的静态成员访问语法必须使用双冒号 `::`。绝不可能出现单独一个冒号 `:` 并且诡异地和指针操作符 `->` 混用的语法结构。

Question 81

Which among the following is correct definition for static member functions?

- A. Functions created to allocate constant values to each object
- **B. Functions made to maintain single copy of member functions for all objects**
- C. Functions created to define the static members
- D. Functions made to manipulate static programs

解析：正确选项是 **B**。注意：在底层物理内存上，无论是普通函数还是静态函数，所有对象的函数代码本来就只有一份。然而，这是各大考研/计算机基础题库中的经典题：出题人意在强调静态函数体现了“大家共有、不属于某一个具体对象”的特性。它属于整个类的抽象级别（Class-level），是独立于具体实例的单一公共方法。

Question 90

The keyword static is used _____

- A. With declaration inside class and with definition outside the class
- **B. With declaration inside class and not with definition outside the class**
- C. With declaration and definition wherever done
- D. With each call to the member function

解析：正确选项是 **B**。这是 C++ 的硬性语法规定。当你在类的外部去单独实现（定义）一个静态成员（无论是静态数据变量的内存分配，还是静态函数的实现）时，**绝对不能**在前面再加 `static` 关键字，否则会导致链接错误。`static` 只能出现在类内部的声明处。

Question 93

What will be the output if all necessary code is included?

```
void test (Object &y) { y = "It␣is␣a␣string"; }
void main() {
    Object x ;
    test (x);
    cout<<x;
}
```

- A. Run time error
- B. Compile time error
- C. Null
- **D. It is a string**

解析：正确选项是 **D**。在 `test` 函数中，形参使用的是对象的引用（`Object &y`）。按引用传递意味着 `y` 只是外部对象 `x` 的一个别名。因此，在函数内部对 `y` 进行赋值，本质上直接覆写了外部 `x` 的内存数据。最终输出修改后的字符串。

Question 100

Which error will be produced if a local object is returned by reference outside a function?

- A. Out of memory error
- **B. Run time error**
- C. Compile time error
- D. No error

解析：正确选项是 **B**。这是一个非常致命且经典的“悬空引用（Dangling Reference）”错误。局部对象存在于函数的栈内存中，一旦函数执行结束，这块内存立刻被系统回收。如果返回了它的引用，外部调用者将拿到一个“指向已销毁内存”的毒指针，对其访问极易导致 `Segmentation fault` 等运行时崩溃（Run time error）。

Question 102

Can we return an array of objects?

- A. Yes, always
- B. Yes, only if objects are having same values

- C. No, because objects contain many other values
- **D. No, because objects are single entity (arrays are not a single entity)**

解析：正确选项是 **D**。在 C/C++ 底层机制中，原生数组（Native Array）不支持整体按值复制，因此不能作为函数的返回值直接返回。如果要返回一组数据，我们只能返回指向数组首元素的指针，或者更现代的做法是使用 `std::vector` 或 `std::array` 将数组包装为单一的对象实体后再返回。

Question 103

Which among the following is true?

- **A. Two objects can point to the same memory location**
- B. Two objects can never point to the same memory location
- C. Objects not allowed to point at a location already occupied
- D. Objects can’t point to any address

解析：正确选项是 **A**。通过指针操作或引用传递，C++ 完全允许让不同的对象（或指针对象）指向堆区或栈区中的同一块物理内存地址。这也是导致“浅拷贝引发内存重复释放（Double Free）”这一经典 Bug 的根本原因。

Question 108

How the argument passed to a function get initialized?

- **A. Assigned using copy constructor at time of passing**
- B. Copied directly
- C. Uses addresses always
- D. Doesn’t get initialized

解析：正确选项是 **A**。当对象作为参数进行“按值传递（Pass by Value）”时，系统绝不是简单粗暴地做底层字节复制。相反，系统在开辟函数栈区时，会正规地调用该类的“拷贝构造函数（Copy Constructor）”来精心构建并初始化这个局部的形参对象。

Question 111

In copy constructor definition, if non const values are accepted only _____

- A. Only const objects will be accepted
- **B. Only non -const objects are accepted**
- C. Only const members will not get copied

- D. Compiler generates an error

解析：正确选项是 **B**。拷贝构造函数的参数通常会写成 `const ClassName&`。如果你没有加 `const` 修饰（即 `ClassName(ClassName& obj)`），这就表示你的函数具有修改源对象的潜力。此时，编译器基于类型安全，将**绝对不允许**你把一个只读的 `const` 对象传给它。因此它变得只能接受非常量对象（non-const objects）。

Question 112

Use of assignment operator _____

- **A. Changes its use, when used at declaration and in normal assignment**
- B. Doesn't changes its use, whatever the syntax might be
- C. Assignment takes place in declaration and assignment syntax
- D. Doesn't work in normal syntax, but only with declaration

解析：正确选项是 **A**。赋值符号 `=` 在 C++ 中扮演双重角色，根据上下文改变其底层行为：如果在对象刚刚声明创建时使用（例如 `A obj2 = obj1;`），它实际上调用的是**拷贝构造函数**；如果在对象早已创建完毕之后的某个语句中使用（例如 `obj2 = obj1;`），它调用的则是重载的**赋值运算符 (`operator=`)**。

Question 117

Which among the following is true?

- A. We can use direct assignment for any object
- B. We can use direct assignment only for different class objects
- C. We must not use direct assignment
- **D. We can use direct assignment to same class objects**

解析：正确选项是 **D**。在 C++ 中，即使你没有手动编写任何赋值逻辑，编译器也会自动为每一个类隐式生成一个默认的赋值运算符重载。这意味着，只要属于相同的类，这两个不同实例对象之间就可以直接使用等于号（`=`）进行互相赋值（默认执行逐成员的浅拷贝）。

Question 119

What is the size of an object pointer?

- **A. Equal to size of any usual pointer**
- B. Equal to size of sum of all the members of object
- C. Equal to size of maximum sized member of object

- D. Equal to size of void

解析：正确选项是 **A**。无论一个类包含多少成员，无论该类的实例对象占用多庞大的内存空间，**指向该对象的指针仅仅是一个存放内存地址的变量**。因此，对象指针的大小永远等于当前计算机架构下普通指针的大小（例如 32 位操作系统下为 4 字节，64 位系统下为 8 字节）。

Question 127

An object’s this pointer _____

- A. Isn’t part of class
- B. Isn’t part of program
- C. Isn’t part of compiler
- **D. Isn’t part of object itself**

解析：正确选项是 **D**。`this` 指针的本质是编译器在调用类的非静态成员函数时，在幕后隐式传递的**第一个函数参数**（用于告诉函数当前是在操作哪一个对象）。它根本不存储在对象的物理内存布局中。因此对对象执行 `sizeof()`，其结果绝对不包含 `this` 指针的体积。

Question 130

Which is the correct interpretation of the member function call from an object, object.function(parameter);

- A. object.function(&this, parameter)
- B. object(&function,parameter)
- C. function(&object,&parameter)
- **D. function(&object,parameter)**

解析：正确选项是 **D**。这是 C++ 面向对象机制最底层的真相：在编译器眼中，其实根本没有所谓的“成员函数”。当你在代码里写出 `object.function(arg)` 时，编译器在底层会将其重写并转化为普通的全局函数调用：`function(&object, arg)`。它悄悄地把调用者的内存地址（即 `&object`）作为第一个隐藏参数传了进去，这个隐藏参数在函数内部接收的名字，就叫做 `this` 指针。

Question 132

Which among the following is true? (About this pointer)

- A. This pointer can be used to guard against any kind of reference
- **B. This pointer can be used to guard against self-reference**

- C. This pointer can be used to guard from other pointers
- D. This pointer can be used to guard from parameter referencing

解析：正确选项是 **B**。在重载赋值运算符（`operator=`）时，我们必须要防范一种极度危险的操作："自我赋值（Self-assignment）"，比如 `obj = obj;`。因为赋值操作通常会先释放自己原有的旧内存，如果是自我赋值，就会把自己的新数据也连同销毁。因此，标准的做法是利用 `this` 指针进行防御性检查：`if (this != &other) { ... }`。

Question 136

This pointer can be used directly to _____

- **A. To manipulate self-referential data structures**
- B. To manipulate any reference to pointers to member functions
- C. To manipulate class references
- D. To manipulate and disable any use of pointers

解析：正确选项是 **A**。`this` 指针最常见的直接运用之一，就是在成员函数末尾返回 `*this`。这使得该函数返回了当前对象本身的引用，从而允许程序员像拼图一样进行极其优雅的"链式调用（Method Chaining）"，例如：`obj.setX(1).setY(2).print();`。

Question 137

Which is the correct syntax for declaring the type of this in a member function?

- A. classType [cv-qualifier-list] *const this;
- B. classType const[cv-qualifier-list] *this;
- C.[cv-qualifier-list]*const classType this;
- **D.[cv-qualifier-list] classType *const this;**

解析：正确选项是 **D**。理解这一点的关键是：`this` 就像你的身份证，它从你"出生"（进入函数）那一刻起就绑定在你身上，**绝对不允许重新指向其他对象**。因此，必须将 `const` 放在 `*` 号的右边，声明它是一个"常量指针（Constant Pointer）"。若外层函数还有 `const` 修饰（即 `cv-qualifier-list`），则指向的数据也会变为不可变。

Question 139

If a constructors should be capable of creating objects without argument and with arguments, which is a good alternative for this purpose?

- A. Use zero argument constructor

- B. Use constructor with one parameter
- **C. Use constructor with all default arguments**
- D. Use default constructor

解析：正确选项是 **C**。在 C++ 中，如果我们提供了一个所有参数都带有默认值的构造函数（例如 `ClassName(int a = 0, int b = 0);`），它就能同时扮演两个角色：当你传参时它是带参构造，当你不传参时它就是无参（默认）构造。这能减少代码冗余。

Question 142

If the constructors are overloaded by using the default arguments, which problem may arise?

- **A. The constructors might have all the same arguments except the default arguments**
- B. The constructors might have same return type
- C. The constructors might have same number of arguments
- D. The constructors can't be overloaded with respect to default arguments

解析：正确选项是 **A**。滥用带默认参数的构造函数极其容易引发**二义性（Ambiguity）冲突**。如果类中同时定义了 `A()` 和 `A(int x = 0)`，当用户编写 `A obj;` 试图实例化对象时，编译器将彻底精神分裂：它不知道你到底是想调用无参构造函数，还是想调用那个带默认值的函数，最终会导致直接报编译错误。

Question 145

Which constructor definition will produce a compile time error?

- A. className(int x=0);
- B. className(char c);
- **C. className(int x=0, char c);**
- D. className(char c, int x=0);

解析：正确选项是 **C**。这是 C++ 形参默认值的铁律：**函数的默认参数必须严格遵循“从右向左”连续填充的原则**。你绝对不能把一个带有默认值的参数放在一个没有默认值的参数的左边。如果不遵守此规则，编译器在匹配参数时将引发灾难性的混乱。

Question 147

Which is the correct statement for default constructors?

- A. The constructors with all the default arguments
- B. The constructors with all the null and zero values
- C. The constructors which can’t be defined by programmer
- **D. The constructors with zero arguments**

解析：正确选项是 **D**。C++ 标准术语中的“默认构造函数（Default Constructor）”，狭义上就是专指那个**不接收任何参数**的构造函数（包括你手动写了空参数的，或者所有参数都提供默认值的，或者是编译器自动生成的）。

Question 150

What happens when new fails?

- A. Returns zero always
- B. Throws an exception always
- **C. Either throws an exception or returns zero**
- D. Terminates the program

解析：正确选项是 **C**。在现代 C++ 标准中，当堆内存耗尽导致 `new` 失败时，默认行为是抛出 `std::bad_alloc` 异常。但如果你使用了特殊的“不抛异常”版本，即 `new (std::nothrow) Type;`，那么在分配失败时，它就不会抛异常，而是优雅地返回一个空指针 `nullptr` (即 0)。

Question 160

If delete is used to delete an object which was not allocated using new _____

- A. Then out of memory error arises
- B. Then unreachable code error arises
- **C. Then unpredictable errors may arise**
- D. Then undefined variable error arises

解析：正确选项是 **C**。这是一个绝对禁忌的严重错误！`new` / `delete` 必须在堆（Heap）内存上成对使用。如果你试图对一个分配在栈（Stack）上或全局数据区的对象地址调用 `delete`，底层的内存管理器会因找不到合法的堆内存元数据块而崩溃，这在 C++ 中属于典型的未定义行为（Undefined Behavior, UB），会导致难以预测的恐怖后果。

Question 162

When delete operator is used _____ (If object has a destructor)

- A. Object destructor is called after deallocation
- **B. Object destructor is called before deallocation**
- C. Object destructor is not used
- D. Object destructor can be called anytime during destruction

解析：正确选项是 **B**。与 `new` 操作符相反，`delete` 的执行过程严格分为两步：第一步，先**就地调用对象的析构函数**，让对象有尊严地清理掉自己手中的资源；第二步，再调用全局的 `operator delete`，将这块物理内存彻底交还给操作系统（deallocation）。

Question 163

If delete is applied to an object whose l-value is modifiable, then _____ after the object is deleted.

- A. Its value is defined as null
- B. Its value is defined as void
- C. Its value is defined as 0
- **D. Its value is undefined**

解析：正确选项是 **D**。对指针执行 `delete` 操作，仅仅是把指针指向的那块内存还给了操作系统，**它并不会自动把你的指针变量清空为 `nullptr`**。操作结束后，这个指针里仍然保留着原来的物理地址，这就产生了一个极度危险的“悬挂指针（Dangling Pointer）”。任何后续对它的解引用，后果都是未定义（Undefined）的。这也是为什么我们在 delete 之后强烈建议手动写上 `ptr = nullptr;` 的原因。

Question 164

Which is the correct syntax to delete an array of objects?

- **A. delete[] objectName;**
- B. delete * objectName;
- C. objectName[] delete;
- D. delete objectName[];

解析：正确选项是 **A**。使用 `new Type[N]` 申请的对象数组，在释放时必须配套使用带有方括号的 `delete[]`。方括号相当于给编译器的一个明显信号：不要只析构第一个元素，请去读取隐藏的数组长度信息，并调用数组中每一个元素的析构函数！若漏掉 `[]` 会引发严重的内存/资源泄漏。

Question 165

The delete operator _____

- **A. Invokes function operator delete**
- B. Invokes function defined by user to delete
- C. Invokes function defined in global scope to delete object
- D. Doesn't invoke any function

解析：正确选项是 **A**。`delete` 是 C++ 的一个关键字级别的操作符，它在完成析构后，底层必然会调用配套的 `operator delete` 函数来实现物理内存的释放。虽然开发者可以重载自己的 `operator delete`，但若不显式指定，最终也会调用标准库提供的版本。

Question 167

What are inbuilt classes?

- **A. The predefined classes in a language**
- B. The classes that are defined by the user
- C. The classes which are meant to be modified by the user
- D. The classes which can't be used by the user

解析：正确选项是 **A**。"内置类（Inbuilt Classes）"是指由编程语言核心或者标准库预先定义好、开箱即用的类。在 C++ 中，最典型的内置类代表就是标准模板库（STL）中提供的 `std::string`，以及各类 I/O 流（如 `std::istream`）。

Question 169

What doesn't inbuilt classes contain?

- A. Function prototype
- B. Function declaration
- C. Function definitions
- **D. Objects**

解析：正确选项是 **D**。这道题重申了 OOP 的基本哲学理念：类（Class）永远只是描绘蓝图和设计规范的数据结构（里面包含函数声明和定义）。它本身是抽象的，里面**绝对不会直接包含实体化的对象（Objects）**。对象是运行时通过这份蓝图才被建造出来的。

Question 171

What is an array of objects?

- **A. An array of instances of class represented by single name**
- B. An array of instances of class represented by more than one name
- C. An array of instances which have more than 2 instances
- D. An array of instances which have different types

解析：正确选项是 **A**。对象数组本质上就是内存中连续存放的、属于同一个类的多个实例（Instances）。为了方便索引和管理，它们在代码中共享同一个名字（Single name），开发者通过下标（例如 `students[0]`, `students[1]`）来分别定位每个独立的对象。