1. Which object in the following code is bound to many names?

***int \*p = new int;***

***int \*q = p;***

🡪 **Object created by new int.**

2. What are the names of the objects which is bound to many names?

🡪 **\*p and \*q.**

3. Given the following C++ code:

***template <class myType>***

***myType GetMax (myType a, myType b) {***

***return (a>b?a:b);***

***}***

Which concept does the above code illustrate ?

🡪 **Polymorphism**.

4. Given the following code written in C:

***x = a \* b;***

What is the binding time of variable x and its type?

🡪 **Programming Time - Thời gian lập trình.**

5. Let x in the above code be a global variable. What is the binding time of x and its absolute address in main memory?

🡪 **Loading time.** (Correct, loading time is the period when a loader loads an executable file from secondary memory into main memory. It is up to the situation of main memory that the executable program will be allocated into appropriate memory. After the allocation, all global variable will have the absolute addresses and the binding between a global variable and its absolute address occurs.)

6. Which time does the binding between \* and its meaning (multiplication operator) happen in programming language C?

🡪 Language design time. (Correct, in imperative programming languages like C, the functions whose name is symbol like \* are determined in language specification. Therefore, the binding between \* and its meaning happens in the language design time.)

7. Given the following C fragment:

***int x;***

***void foo(int y) {***

***static int z;***

***int \* t = malloc(sizeof(int));***

***...***

***}***

Choose the WRONG statement?

🡪 **t is allocated in heap memory.** (Yes, t is a variable in pointer type. It is allocated in stack memory as it is a local variable of function foo. The object to which p points is allocated in heap memory)

8. Given the following C fragment:

***int x;***

***void foo(int y) {***

***static int z;***

***int \* t = malloc(sizeof(int));***

***...***

***}***

Choose the WRONG statement?

A. The lifetime of x is the same as the lifetime of the whole program.

B. The lifetime of y is the same as the lifetime of function foo.

(No, this is correct statement. y is the parameter of function foo so it is considered as a local variable which is allocated in stack memory and lives when function foo executes)

**C. The lifetime of z is the same as the lifetime of function foo.**

**(Yes, although z is a local variable, its lifetime is the same as the lifetime of the whole program as the variable is allocated in static memory)**

D. The lifetime of t is the same as the lifetime of function foo

(No, this is correct statement. t is a local variable of pointer type so it lives just when the function runs. The lifetime of the object t points to however is not the same as the lifetime of function foo.)

9. Given the following C code:

//position 1

int \* foo() {

// position 2

x[0] = 1;

return x;

}

Which declaration of x and at which position can cause a runtime error (dangling reference or garbage)?

A. int x[10]; //position 1

(Wrong, an object declared at position 1 (i.e global) will be allocated in static area so its lifetime is the same as the lifetime of the program; the object cannot be garbage. The above code also causes the dangling reference.)

**B. int x[10]; //position 2**

**(Correct, an object declared at position 2(i.e. local) will be allocated in stack area. Its lifetime is the same as the lifetime of the enclosed function (i.e. foo); it is allocated when the function runs and destroyed when the function is terminated. The instruction "return x" gives the reference to the object to some variable outside the functions. The reference to the object still exists after the object is killed when the function is terminated. This scenario causes "dangling reference")**

C. static int x[10]; // position 2

(Wrong, an object declared with keyword "static" will be allocated in static area although it is only referred inside the enclosed function.Its lifetime is the same as the lifetime of the program; it always be referred when the function foo is invoked so it cannot be a garbage. It is not destroyed when the program runs so it cannot cause a dancing reference.)

D. int\* x = malloc(10\*sizeof(int)); // position 2

(Wrong, an object created by malloc will be allocated in heap area. It is not destroyed so it does not cause "dangling reference". It might be a garbage but the pointer is returned to outside. The error of "garbage" might be caused by the outside code, not by the above code.)