Assignment 07/08/2024

Author: ThanhTH10

Date: 8/8/2024

1. What is dangling pointer? Implications of dangling pointers

***A dangling pointer*** is a pointer in programming that points to memory that has been deallocated or is no longer valid. This is a common and dangerous error in C and C++ programming.

***Main causes of dangling pointers:***

1. Freeing memory that a pointer is pointing to without updating the pointer.
2. Returning the address of a local variable from a function.
3. Using a pointer after the object it points to has been destroyed.

Example:

int\* createDanglingPointer() {

    int x = 5;

    return &x; // Returning address of local variable

}

int main() {

    int\* ptr = createDanglingPointer();

    // ptr is now a dangling pointer

    \*ptr = 10; // Undefined behavior

    return 0;

}

***Implications of dangling pointers:***

1. Undefined behavior: Accessing invalid memory can lead to unpredictable results.
2. Security vulnerabilities: Can lead to serious security flaws, allowing attackers to execute malicious code.
3. Program crashes: Accessing freed memory can cause segmentation faults or program crashes.
4. Hard-to-detect bugs: Dangling pointers may not cause immediate errors, making debugging difficult.
5. Memory leaks: If the only pointer to a memory location becomes dangling, it can lead to memory leaks.
6. Data corruption: Modifying data through a dangling pointer can corrupt data in other parts of the program.

***To avoid dangling pointers:***

1. Set pointers to NULL after freeing memory.
2. Use smart pointers in modern C++.
3. Avoid returning addresses of local variables from functions.
4. Be careful when managing object lifecycles.

2. What all issues with shallow copying approach

***The main issues with the shallow copying approach in C++ are:***

1. **Dangling Pointers:** When you perform a shallow copy of an object that contains pointers, both the original object and the copy will have pointers pointing to the same memory location. If the original object's pointer is later deallocated or goes out of scope, the copy's pointer becomes a dangling pointer, leading to undefined behavior.
2. **Data Inconsistency:** With a shallow copy, if the original object's data members are modified, the copy will also be affected, and vice versa. This can lead to data inconsistency and unexpected behavior.
3. **Memory Leaks:** If the original object's dynamically allocated memory is not properly deallocated, a shallow copy will also have a pointer to that memory, leading to memory leaks.
4. **Lack of Isolation:** Shallow copies do not provide isolation between the original object and the copy. Changes made to one object can inadvertently affect the other, which can be problematic in complex systems.
5. **Incomplete Copying:** Shallow copies only copy the values of the object's data members, but not the values pointed to by those data members. This means that the copy does not truly represent a complete and independent copy of the original object.