VNUHCM-UNIVERSITY OF SCIENCE

FACULTY OF INFORMATION TECHNOLOGY

CSC10003 – OBJECT-ORIENTED PROGRAMMING

Lab 5: Assignment 02 Question 1 & 2

Lecturer
Mr. Nguyễn Lê Hoàng Dũng
Mr. Hồ Tuấn Thanh

Class
23CLC08

Students
23127255 - Nguyễn Thọ Tài

25/11/2024

Summary

Original Code

```
-5      #include <iostream>
-4      using namespace std;
-3
-2      #include "cstring"
-1
1       class A{
2               char *m_s;
3       public:
4               A() { m_s = strdup("default"); }
5               A(char *s) { m_s = s; }
6               virtual void prepare() { cout << "A "; }
7               void display() {
8                       prepare();
9                       cout << m_s << endl;
10              }
11      };
12
13      class B : public A{
14      public:
15              B(char *s) : A(s) {}
16              B(const B &b) {}
17              void prepare() { cout << "B "; }
18      };
19
20      void foo(A *obj1, A obj2) {
21              obj1->display();
22              obj2.display();
23      }
24
25      int main() {
26              B obj1("text");
27              A *obj2 = new B(obj1);
28              foo(&obj1, *obj2);
29              return 0;
30      }
```

Question 1: What is printed to the console? Give an brief explanation.

1. Ouput

```
B text
A default
```

2. Explanation

```
26      B obj1("text")
```

In line 26, object obj1: B was created as an object saved in stack using constructor with input `char *` -> A(char *) got called then B(char *) -> obj1's **m_s** is `"text"`

```
27      A *obj2 = new B(obj)
```

In line 27, object `obj2: A *` was allocated by copying obj1 using copy constructor of B. But the copy constructor of B override the default one and do nothing (no calling A's copy constructor + no body) -> A's default constructor got called. -> obj2's **m_s** is `"default"`

```
20      void foo(A *obj1, A obj2) {
21              obj1->display();
```

```
22          obj2.display();
...
26          B obj1("text");
27          A *obj2 = new B(obj1);
28          foo(&obj1, *obj2);
```

In line 28, function `foo` got called with obj1 got referenced and obj2 got dereferenced before passing to `foo`
In line 20, obj1 is a pointer; obj2 is a normal type. So as a result, obj1 got passed by reference but obj2 got passed by value.

In function `foo`, obj1 is still the same object from previous scope while obj2 is a copy of the original object.
Thus, obj1 still keep its polymorphism and obj2 lost its polymorphism.

- obj1 is now an `A` pointer point to the original `B` object, so calling `obj1->display()` output `"B text"`
- obj2 is now an `A` object, so calling `obj2.display()` output `"A default"` to the console.

Question 2: Identify the memory issues in the above program and the correct them.

1. Memory issue

1.1 Object A does not have a method to free allocated memory (deconstructor)

```
1     class A{
2             char *m_s;
3     public:
4             A() { m_s = strdup("default"); }
5             A(char *s) { m_s = s; }
6             virtual void prepare() { cout << "A "; }
7             void display() {
8                     prepare();
9                     cout << m_s << endl;
10            }
11    };
```

Solutions: Add a deconstructor to free the allocated `m_s`. Optionally, making the deconstructor virtual so derived class B can delete its allocated properties.

```
1     class A{
2             char *m_s;
3     public:
4             A() { m_s = strdup("default"); }
5             ~A() { delete[] m_s; }
...
12    };
```

1.2 Line 27, allocated obj2 does not get freed.

```
27    A *obj2 = new B(obj1);
28    foo(&obj1, *obj2);
29    return 0;
```

Solutions: Free obj2 when the scope is about to get freed.

```
27    A *obj2 = new B(obj1);
28    foo(&obj1, *obj2);
29    delete obj2;
30    return 0;
```

1.3 Class A got shallow copy leading to its copy deleting the allocated properties

This leading to multiple delete of the same pointer **Solutions:** Add a deep copy constructor to class A

```
1       class A{
2               char *m_s;
3       public:
4               A() { m_s = strdup("default"); }
5               A(const A &other) { m_s = strdup(other.m_s); }
```

1.4 Line 5, m_s get shallow-copied. m_s might not point to a dynamic array, m_s is not in Heap

```
1       class A{
2               char *m_s;
3       public:
4               A() { m_s = strdup("default"); }
5               A(char *s) { m_s = s; }
```

Either derived class of A pointing to the same memory and multiple deletes of the same pointer or m_s is not in Heap (char [100] in heap) **Solutions:** Allocate a copy of m_s

```
1       class A{
2               char *m_s;
3       public:
4               A() { m_s = strdup("default"); }
5               A(char *s) { m_s = strdup(s); }
```

2. Final code

```
#include <iostream>
using namespace std;

#include "cstring"

class A{
        char *m_s;
public:
        A() { m_s = strdup("default"); }
        ~A() { delete[] m_s; }
        A(char *s) { m_s = strdup(s); }
        A(const A &other) { m_s = strdup(other.m_s); }
        virtual void prepare() { cout << "A "; }
        void display() {
                prepare();
                cout << m_s << endl;
        }
};

class B : public A{
public:
        B(char *s) : A(s) {}
        B(const B &b) {}
        void prepare() { cout << "B "; }
};

void foo(A *obj1, A obj2) {
        obj1->display();
        obj2.display();
}

int main() {
        B obj1("text");
        A *obj2 = new B(obj1);
        foo(&obj1, *obj2);
        delete obj2;
        return 0;
}
```