VNUHCM-UNIVERSITY OF SCIENCE

FACULTY OF INFORMATION TECHNOLOGY

CSC10003 – OBJECT-ORIENTED PROGRAMMING

Assignment 02

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

Class
23CLC08

Students
23127226 – Nguyễn Đăng Minh

November 26th, 2024

Contents

Here is the image of the code:

```
#include <iostream>
using namespace std;
class A {
private:
      char *m_s;
public:
      A() { m_s = strdup("default"); }
      A(char *s) { m_s = 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();
}
void main() {
      B obj1("text");
      A *obj2 = new B(obj1);
      foo(&obj1, *obj2);
}
```

Figure 1: The code to analyze

1 Question 1

Output of the above code is:

B text

A default

Explanation:

First line: B text

We initialize obj1 as B object and assign it with string "text", so when we pass its address to the foo function, it will call the display function that inherits from A .The prepare function overrides from class A and therefore prints "B ". The m_s equals to "text" as we have initialized at first.

Second line: A default

Even though we have a pointer obj2 of A type and it points to a B object that copys from obj1 .But since we haven't written any codes in B copy constructor, so it will call the default constructor from A which gives m_s = "default". When we passed it into the foo function, we didn't pass it as a pointer but we cast it into an A object. So the prepare() function is the A's function instead of being overridden from B. That's why it prints "A " and not "B ". In conclusion, the result that obj2 prints out is "A default".

2 Question 2

1) In the constructor A(char *s) of class A:

It performs shallow copy as assign the pointer m_s to be equal to s. Shallow copy is dangerous because they are both pointing to a same memory. We need to fix it by changing to deep copy. The code is:

```
m_s = new char[strlen(s) + 1];
for (int i = 0; i < strlen(s); ++i) m_s[i] = s[i];
m_s[strlen(s)] = '\0';
```

2) In the copy constructor of class B:

In class B, there is a copy constructor and it hasn't done anything. And since B doesn't have access to m_s pointer, so we need to define a copy constructor in A so B can use it. Of course, we'll do a deep copy in copy constructor. The code is:

```
A(const A &other) {
    m_s = new char[strlen(other.m_s) + 1];
    for (int i = 0; i < strlen(other.m_s); ++i) m_s[i] = other.m_s[i];
    m_s[strlen(other.m_s)] = '\0';

}
```

And so we fix the copy constructor in class B as followed:

```
B(const B &b) : A(b) {}
```

3) No (virtual) destructor in class A:

A is the superclass so it also requires a virtual destructor. We need destructor to delete allocated memory of m_s, we need the destructor to be virtual to properly delete subclasses of A too. The code is:

```
virtual  A() { delete m_s; }
```

4) Finally, in the main function:

In the main function, we allocate a new memory of B object, and we haven't free the memory yet which leads to memory leak. So we need to add:

```
delete obj2;
```

3 Adding input and output operators for class A and B

The updated code is in 23127226_Ex02.cpp.

```
friend istream &operator » (istream &in, A &a);
friend ostream &operator « (ostream &out, const A &a);
friend istream &operator » (istream &in, B &b);
friend ostream &operator « (ostream &out, const B &b);
```