Question: Consider the following pieces of code. For each of them, indicate if they will run correctly. If not, what kind of error would they generate (compile time, runtime, or logical). State the reason the programs would not run correctly, if applicable. If the programs would run correctly, what would be the output?

### Code 1
```cpp
#include <iostream>
using namespace std;

class Test
{
private:
    int num1;
    int num2;
public:
    Test(int x, int y);
    static void print();
};
Test::Test(int x, int y)
{
    num1=x;
    num2=y;
}
void Test::print()
{
    cout << num1 << num2 << endl;
}

int main()
{
    Test t(10,9);
    t.print();
    return 0;
}
```

---

<table>
<thead>
<tr>
<th>Run Correctly? (Yes/No)</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>What kind of error(s)?</td>
<td>Compile Time</td>
</tr>
<tr>
<td>Reason(s):</td>
<td>Trying to access non-static variables from static method</td>
</tr>
<tr>
<td>Output:</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

### Code 2
```cpp
#include <iostream>
using namespace std;

class Room
{
    const int max_no_of_rooms;
    static int no_of_rooms;
public:
    Room();
    static void print();
};
Room::Room()
{
    no_of_rooms = 1;
}
void Room::print()
{
    cout << no_of_rooms << endl;
}

int main()
{
    Room r;
    r.print();
    return 0;
}
```

---

<table>
<thead>
<tr>
<th>Run Correctly? (Yes/No)</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>What kind of error(s)?</td>
<td>Link Time (compile Time in VC++; see reason below)</td>
</tr>
<tr>
<td>Reason(s):</td>
<td>variable no_of_rooms used in print method without prior declaration (illegal reference!)</td>
</tr>
<tr>
<td>Note that VC++ will also require you to initialize the const int max_no_of_rooms in the constructor initialization list, so your code will not compile in VC++. We will accept both answers.</td>
<td></td>
</tr>
<tr>
<td>Output:</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>
```cpp
#include <iostream>
using namespace std;
const int MAX_NO=100;

class Movie
{
private:
    char *name;
    char *actor;
public:
    Movie();
    Movie(char*, char*);
    Movie(const Movie& m);
    ~Movie();
};
Movie::~Movie()
{
    delete [] name;
    delete [] actor;
}
Movie::Movie()
{
    name = new char[MAX_NO];
    actor = new char[MAX_NO];
    cout << "Default c’tor" << endl;
}
Movie::Movie(char* n, char* a)
{
    name = new char[MAX_NO];
    actor = new char[MAX_NO];
    strcpy(name, n);
    strcpy(actor, a);
    cout << "Just a c’tor" << endl;
}
Movie::Movie(const Movie& m)
{
    name = new char[MAX_NO];
    actor = new char[MAX_NO];
    strcpy(this->name, m.name);
    strcpy(this->actor, m.actor);
    cout << "Copy c’tor" << endl;
}
int main()
{
    Movie Lord_of_The_Rings;
    Movie Matrix("Matrix", "Reeves");
    Movie firstCopy(Matrix);
    Movie secondCopy = Matrix;
    Movie thirdCopy;
    thirdCopy = Matrix;
    return 0;
}
```

Run Correctly? (Yes/No) **Yes No**

What kind of error(s)? **Runtime Error**

Reason(s): For the two objects, **Matrix** and **thirdCopy**, their char* data members point to the same memory area. That is, **thirdCopy** is a shallow copy of **Matrix**. The destructor is called for both of these objects once you are exiting from **main()**. This will cause the same memory area to be deleted twice. This is a runtime error and VC++ catches this. However, the version of g++ we are using does not err on this one; perhaps because it delays giving the freed memory back to the operating system.

Output:
On g++ version that we use, you might get the following output but don’t trust it; this is NOT the way things should really happen (we will, however, accept this answer too):

```
Default c’tor
Just a c’tor
Copy c’tor
Copy c’tor
Default c’tor
```