COMP 345 Week 8

h_lai@encs.concordia.ca

Let’s look into it again

In the lab, when I tried to explain the difference between the object created in the stack and heap. I said the following code is **incorrect** (based on the assumption that no copy constructor will be provided, neither by coder nor compiler !!!):

```
class Student {
public:
        Student() = default;                                    // defualt constructor
        Student(const Student &student) = delete;   // Avoiding implicit generation of the copy constructor.
};

Student create() {
        Student s;
        // do something about s
        return s;
}
int main() {
        Student s = create();
        // do something with s
return 0;
}
```

The following code is correct

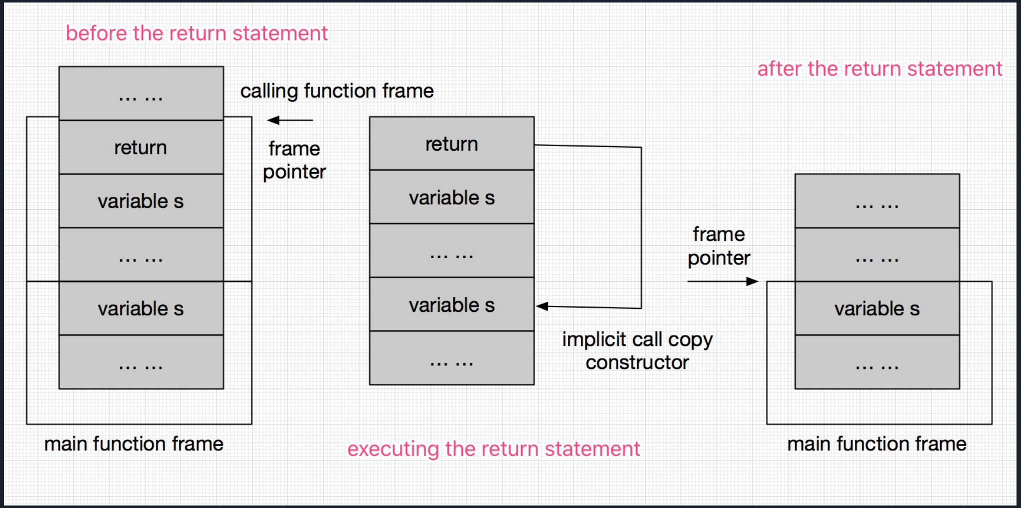
std::string do provides copy constructor

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

string create() {
        string str;             // create a string type variable str on the stack
        str = “hello world”
        return str;             // when program comes here, implicitly called the copy constructor
}
int main() {
        string s = create();  // variable s here is a copy of the str variable in create() function
        cout << s << endl;
        return 0;
}
```

What happened in the stack?

```
string create() {
        string str;
        str = "hello world"
        return str;
}
int main() {
        string s = create();
        cout << s << endl;
        return 0;
}
```



Make the first example worked

```
class Student
{
public:
        Student() = default;                                    // defualt constructor
        ~~Student(const Student &student) = delete;~~       // Avoiding implicit generation of the copy constructor.

};

Student create() {
        Student s;
        // do something with s
        return s;
}
int main() {
        Student s = create();
        // do something with s
        return 0;
}
```

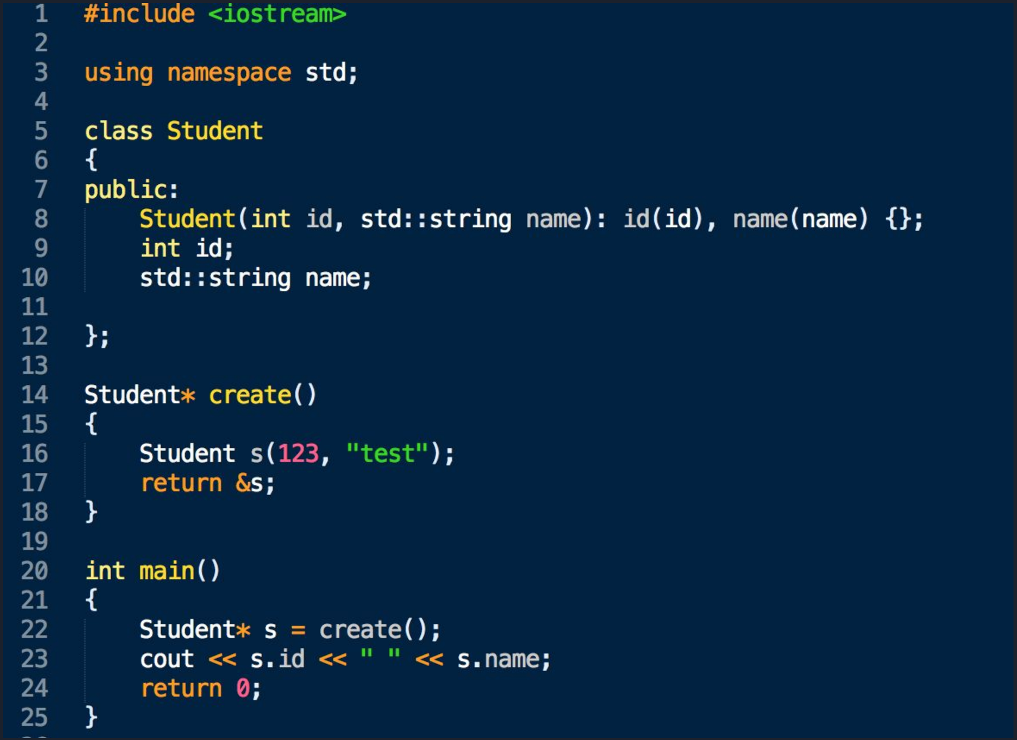
Why it worked ???

If you don’t provide the copy constructor and don’t prevent the compiler to generate it, a copy constructor will be generated by the compiler itself.

The auto generated copy constructor will **shadow copy** all member variables from the old object to a new object.

Some useful links can help to understand:
cpp reference of copy constructor
cpp constructor generation rule
copy constructor generation rule

Another typical incorrect program



```cpp
#include <iostream>

using namespace std;

class Student
{
public:
    Student(int id, std::string name): id(id), name(name) {};
    int id;
    std::string name;

};

Student* create()
{
    Student s(123, "test");
    return &s;
}

int main()
{
    Student* s = create();
    cout << s.id << " " << s.name;
    return 0;
}
```