TA Lecture Notes: Lecture #6 - Functions and Copy Constructor

January 29, 2018 • by *Milica Hadzi-Tanovic*

- Identical to passing parameters into a function, we also have three choices on how memory is used when returning from a function:
  - Return **by value**: the object is copied from the function to the caller function (the memory in function and caller function is different).

```
15  Sphere joinSpheres(const Sphere &s1, const Sphere &s2);
```

  - Return **by reference**: returns the address of the object (*remember*, never return a reference to a stack memory).

```
15  Sphere & joinSpheres(const Sphere &s1, const Sphere &s2);
```

  - Return **by pointer**: also returns the address of the object (*remember*, never return a reference to a stack memory).

```
15  Sphere * joinSpheres(const Sphere &s1, const Sphere &s2);
```

- **Copy Constructor**, as its name says, copies an object based on the existing object. When a non-primitive variable is passed/returned by value, a copy must be made.
  - Automatic Copy Constructor
    - Called if no custom copy constructor is defined.
    - If we are copying an object, it copies all member variables by calling their copy constructors.
    - If we are copying a pointer, it will make a shallow copy by making the variable point to an existing address.
  - Custom Copy Constructor
    - Declared as any other constructor but with special parameter.
    - It is supposed to make a **deep copy** of every member variable.

```
Universe::Universe(const Universe & other)  {
                                          // custom copy ctor of some class Universe
    // Deep copy of an object                // that contains three spheres
    p_ = other.p_;                          // const key word ensures “other” is not changed

   // Deep copy of a pointer
   q_ = new Sphere (other → q_);

    // Deep copy of a reference
    You cannot make a copy of a reference.
    It doesn’t have its own memory.
}
```

- Pass/return by value/reference/pointer tradeoffs

by value		by reference/pointer
Everytime a value is passed or returned the value is copied into a new object.		We are passing around addresses, but the object is the same.
We are not changing the original object, so it is safe.		It is risky because we are changing the original object.