CS11600: Introduction to Computer Programming (C++)

Lecture 9

Svetlozar Nestorov
University of Chicago

Outline

- Data member objects:
  - containment, pointers, references
- An aside on references.
- Class member initialization:
  - containment, references
- Lifetime of objects

Data Member Objects

- An object may be a data member of *another* class .

```
class Drinker {
   Beer favBeer;
… }
```

- Every `Drinker` object *contains* a `Beer` object (`favBeer`).
- Can an object be a data member of its own class?

Data Member Pointers

- A data member can be a pointer to an object (of *any* class).

```
class Drinker {
    Beer favBeer;
    Beer *favCheapBeer;
    Drinker *spouse;
… }
```

- Note that a pointer to `Drinker` is allowed as a data member of `Drinker`.

Data Member References

- A data member can be a reference to an object of *another* class.

```
class Drinker {
   Beer favBeer;
   Beer *favCheapBeer;
   Beer & lastResortBeer;
…}
```

A Beer object contained in the Drinker object.

A pointer to a Beer object, may be NULL.

A reference (implicit pointer) to a Beer object, cannot be NULL.

An Aside on References

- References are variable aliases.

```
int age = 52;
int & myAge = age;
```

- The *lvalue* of `myAge` is the same as the *lvalue* of `age` (i.e., both refer to the same memory location.)
- References *do not* allocate memory; must be initialized with already allocated memory.

Class Member Initialization

- Default constructor is called by default (implicitly).
- Can call another constructor explicitly.

```
class Drinker {
    char *name;
    Beer favBeer;
public:
    Drinker() : favBeer("Bud") {…}
    Drinker(const char *name);
…}
```

- The default `Drinker` constructor calls explicitly the `Beer` constructor.
- The other `Drinker` constructor call implicitly the default `Beer` constructor.

Initialization of Const Data Members

- The only way to initialize const data members is by constructors.
- Remember: No assignments to const objects!

```
class Beer {
  const int taste;
  …
public:
  Beer(char *name, int t) : taste(t) {
  …}
```


Initialization of Reference Members

- The only way to initialize reference data members is by calling constructors explicitly.

```
class Drinker {
    Beer & favBeer;
public:
    Drinker(Beer & b) : favBeer(beer) {
    …}
…}
```

- The `Beer b` object must already exist.

Lifetime of Objects

- Local variable objects: the innermost scope where the variable is defined.

```
{
  …
  Beer bud;
  …
}
```

Lifetime of bud

Beer destructor is called implicitly

Containment

- A contained object exists while the object that contains it exists.

```
class Drinker {
    Beer favBeer;
…}
…
Drinker joe;
```

- When `Drinker` destructor is called (implicitly or explicitly) for `joe`, it calls implicitly `Beer` destructor for `favBeer`.

Object Pointer Data Members

- Lifetime is not directly connected to containing object.
- Two kinds of pointer data members.
- Point to objects created earlier:
  - Not recommended.
- Point to objects created in the constructor:
  - Must be deleted in the destructor.

Object References

- Not connected to lifetime of containing object.
- Danger of dangling references.
- Pros and cons for objects, pointers, and references.