Lecture #2

class composition,constructors,destructors,initializer lists

Constructors: Class Initialization

- A constructor is a special member function that automatically initializes every new variable you create of a new class.
- Just like any C++ function, a constructor can have one or more default parameters.
    - You can have as many constructors as you want, as long as they differ in number and/or type of parameters.
    - If you pass in values for the constructor parameters, it will override the default parameters.
    - You should not have a parameter-less constructor **and** a constructor with all default parameters.
- If you don't define any constructors at all, C++ generates an implicit default constructor for you.
    - The default constructor does not initialize your object's scalar member variables!
- If you declare an array filled with variables of a class, the class must have a default (parameter-less) constructor.

Destructors

- Every class has a **single** destructor.
    - Its job is to de-initialize or destruct a class variable of the class
- If you don't define a destructor, C++ will define an implicit one automatically.
- **Why do we need destructors?**
    - Need to free space allocated for class variables when the program is about to leave its scope

```
class CSNerd {
  public:
  //---------------//
  // Constructors //
  //---------------//
  CSNerd(int PCs, bool usesMac = true) { // default parameter
    m_numPCs = PCs;
    m_macUser = usesMac;
  }

  CSNerd() {
    m_numPCs = 1;
    m_macUser = false;
  }

  //--------------//
  // Destructors //
  //--------------//

  ~CSNerd() {

  }

  int getNerdScore() {
    if(macUser == true)
      return 0;
    return 10 * m_numPCs
  }

  private:
  int m_numPCs;
  bool m_macUser;
}

int main() {
  CSNerd lyn(1, false); // goes to top constructor
  CSNerd ned(5); // goes to top constructor and defaults "usesMac" to true
  CSNerd dave;  // Goes to parameter-less constructor
}
```

Class Composition

- class composition - when a class contains one or more member variables that are objects
- When an outer object contains member objects, C++ automatically adds code to the outer object's constructor to FIRST call the DEFAULT constructors of all the member objects.
- When the outer object destructor is called, the inner objects are destructed at the **END** of the destructor block, in the *reverse* order of construction
- If the outer object destructor is the one automatically created by C++, all it does is ensure that the member variables are properly destructed.
- Auto-added default constructor only initializes class member variables, **not** scalar member variables.
- **Initializer list is mandatory for constructing member variables whose initializers take parameters.**
    - You must always add your initializer list to your actual constructor definition (whether it's defined inside or outside of your class).

```
class Stomach {
  public:
    Stomach(int startGas) { myGas = startGas; }
    void eat() { myGas++; }
  private:
    int myGas;
};

class Brain {
  public:
    Brain(int startIQ) {myIQ = startIQ; }
    void think() { myIQ += 10; }
  private:
    int myIQ;
};

class HungryNerd {
  public:
    HungryNerd(int startingGas): myBelly(startingGas), myBrain(150) {   // initializer list
      myBelly.eat();
```

```cpp
      myBrain.think();
    }
  private:
    Stomach myBelly;
    Brain myBrain;
}

int main() {
  HungryNerd carey;
}

```