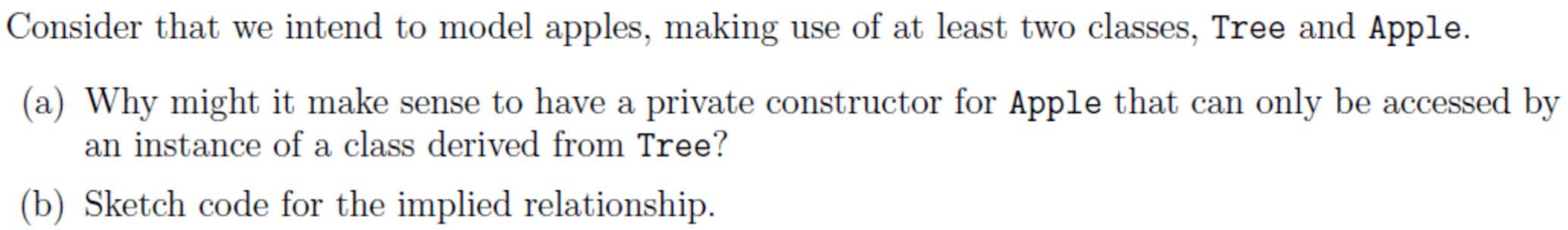
CSCI251/CSCI851 Autumn-2020 Advanced Programming (**LT8**)

Lecture Tutorial 8

From the lab:

- The `Tree` and `Apple` question.



Consider that we intend to model apples, making use of at least two classes, `Tree` and `Apple`.

(a) Why might it make sense to have a private constructor for `Apple` that can only be accessed by an instance of a class derived from `Tree`?

(b) Sketch code for the implied relationship.

- Apple trees are a special kind of tree.
  – So it makes sense to have `appleTree` being a class derived from Tree.
- Apples come from apple trees.
  – So it makes sense that only an `appleTree` can construct `Apple` objects.

- Note that `cout` is an object.
- 2.2 was illustrating associations
  - Likely aggregation `Date` in `Student` and in `Subject`
  - Composition of `Student` in `Subject`.
- A-class is short for Association class.

- You cannot have X inheriting from Y and Y inheriting from X at the same time.

Friends with a cat …

- In the class …

```
friend void showCat(const Cat &);
```

- The function is then …

```
void showCat(const Cat &kit)
{
      cout << "Friend : " << endl;
      cout << "Cat: " << kit.name << " a " << kit.breed << endl;
      cout << "The cat's age is " << kit.age << endl;
      cout << "License fee: $" << kit.licenseFee << endl;
}
```

Deep vs shallow copy: Shallow …

```
class Shallow{
private:
    int *x;
public:
    Shallow(int arg);
    int get() const { return *x; }
    void set(int arg) { *x = arg; }
    void display();
    void displayA();
    ~Shallow(){ delete x;}
};
```

```
Shallow::Shallow(int arg)
{
    x = new int;
    *x = arg;
}

void Shallow::display()
{
    cout << "Contains : " << *x << endl;
}

void Shallow::displayA()
{
    cout << "Contains : " << *x;
    cout << " at : " << x << endl;
}
```

```
int main()
{
    Shallow ob1(1);
    Shallow ob2 = ob1;
    ob1.display();
    ob2.display();

    ob1.set(2);
    ob1.display();
    ob2.display();

    ob1.displayA();
    ob2.displayA();

    return 0;
}
```

A default copy constructor will be used!

Changing ob1 changes ob2 too.

```
Contains : 1
Contains : 1
Contains : 2
Contains : 2
Contains : 2 at : 0x560e414f9e70
Contains : 2 at : 0x560e414f9e70
```

Pointing to the same address

Based on https://owlcation.com/stem/Copy-Constructor-shallow-copy-vs-deep-copy

```
class Deep{
private:
    int *x;
public:
    Deep(int arg);
    int get() const { return *x; }
    void set(int arg) { *x = arg; }
    void display();
    void displayA();
    Deep(const Deep& obj);
    ~Deep(){ delete x;}
};
```

```
Deep::Deep(int arg)
{
    x = new int;
    *x = arg;
}

void Deep::display()
{
    cout << "Contains : " << *x << endl;
}

void Deep::displayA()
{
    cout << "Contains : " << *x;
    cout << " at : " << x << endl;
}

Deep::Deep(const Deep& arg)
{
    x = new int;
    *x = arg.get();
}
```

```
int main()
{
    Deep ob1(1);
    Deep ob2 = ob1;
    ob1.display();
    ob2.display();

    ob1.set(2);
    ob1.display();
    ob2.display();

    ob1.displayA();
    ob2.displayA();

    return 0;
}
```

Provided copy constructor will be used!

Changing ob1 doesn't change ob2.

```
Contains : 1
Contains : 1
Contains : 2
Contains : 1
Contains : 2 at : 0x56162c553e70
Contains : 1 at : 0x56162c553e90
```

Pointing to different addresses