Name: ____

Today's Date: ____

Today's Goals

- Judge potential hash functions.
- Start thinking about subtype polymorphism

Today's Question(s)

What properties should a good hash function have?

Lingering Questions

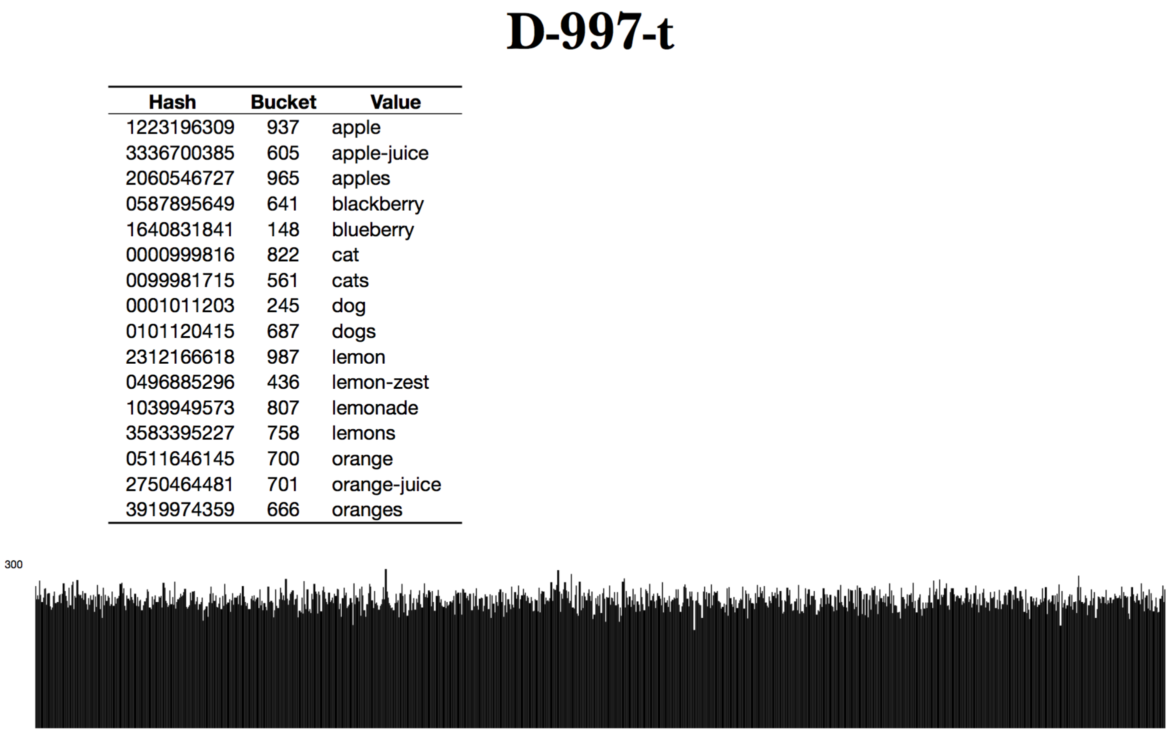
Hash Functions

- What are some good properties?
- How would you pick a hash function?

Sample Terrible Hash Function

```
using uchar = unsigned char;
using uint = unsigned int;
const uint HASH_MULTIPLIER = 32;

uint hash(string str, uint range) {
    uint hashval = 0;
    for (string::iterator i = str.begin(); i != str.end();
        hashval = hashval * HASH_MULTIPLIER + uchar(*i);
 }
    return hashval % range;
}
```

Comparing Hash Functions



D-997-t

Hash	Bucket	Value
1223196309	937	apple
3336700385	605	apple-juice
2060546727	965	apples
0587895649	641	blackberry
1640831841	148	blueberry
0000999816	822	cat
0099981715	561	cats
0001011203	245	dog
0101120415	687	dogs
2312166618	987	lemon
0496885296	436	lemon-zest
1039949573	807	lemonade
3583395227	758	lemons
0511646145	700	orange
2750464481	701	orange-juice
3919974359	666	oranges

Polymorphism

“same name, different forms”

- Overloading - adhoc polymorphism
- Template - static polymorphism
- Class hierarchy - subtype, dynamic polymorphism

Subtype Polymorphism

In C++ (and other object-oriented programming languages), classes are one way to achieve modularity

```
class Cow {
public:
    void speak() const;
};

void Cow::speak() const
{
    cout << "Mooooo" << endl;
}

class Raptor {
public:
    void speak() const;
```

Subtype Polymorphism

How do we make this work?

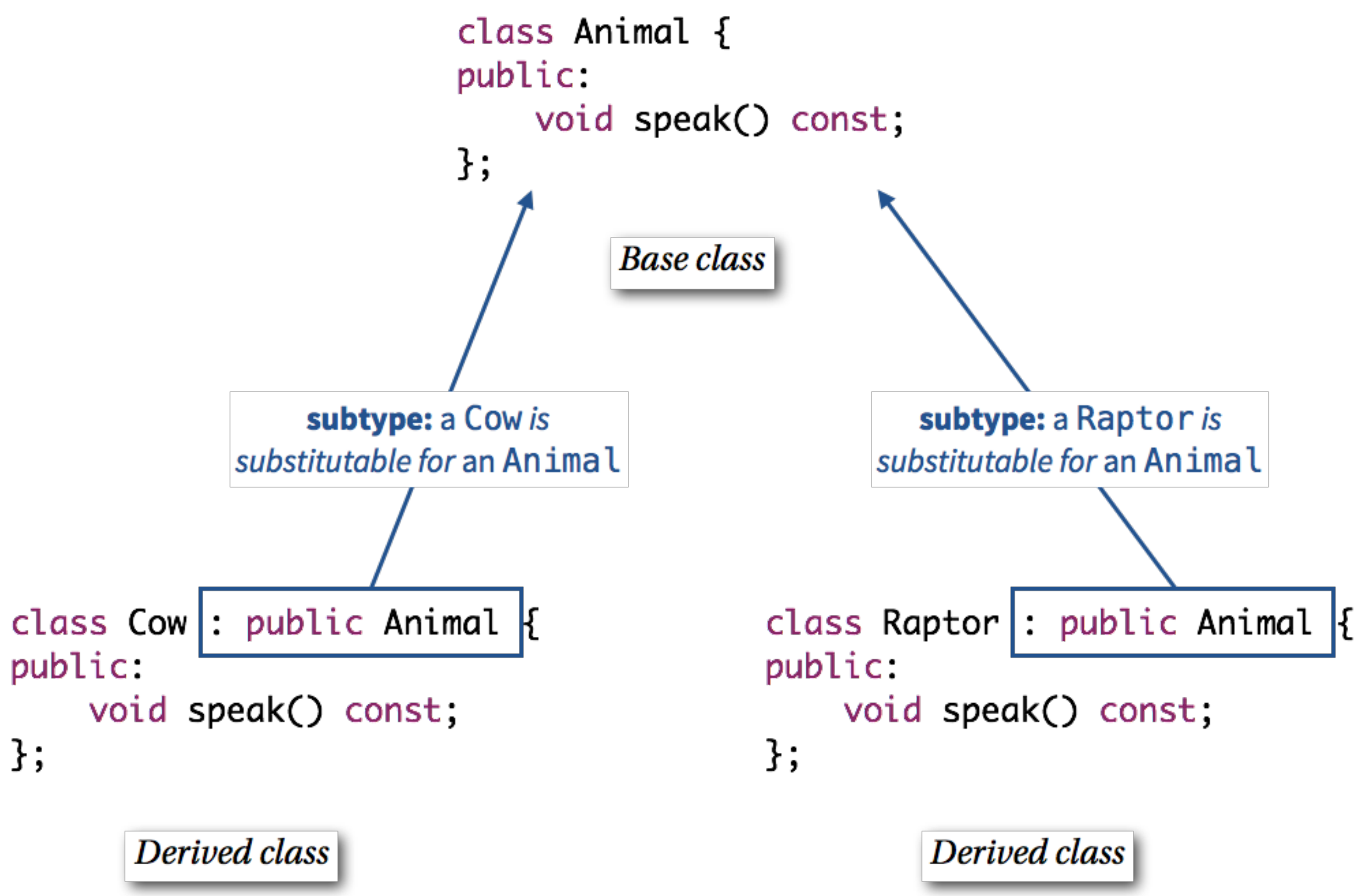
```
void pet(????? creature) {
    creature.speak();
}

{
    Cow bessie;
    Raptor peri;

    pet(bessie);
    pet(peri);
}
```

Liskov Substitution Principle

Derived classes should only expand the capabilities of their base class.

Test: Functions that use pointers or references to base classes must be able to use objects of derived classes without knowing it.

Otherwise, use **overloading** or **templates**.



```cpp
class Animal {
public:
    void speak() const;
};
```

```cpp
class Cow : public Animal {
public:
    void speak() const;
};
```

```cpp
class Raptor : public Animal {
public:
    void speak() const;
};
```

Base class
subtype: a Cow is substitutable for an Animal
subtype: a Raptor is substitutable for an Animal
Derived class
Derived class

Cow and Raptor extend Animal

```
class Animal {
public:
    Animal();
    void speak() const;
private:
    size_t numberOfLegs_;
};

class Cow : public Animal {
public:
    Cow();
    void speak() const;
private:
    double happiness_;
};

Animal::Animal() : numberOfLegs_{4}
{
    // Nothing (else) to do.
}

Cow::Cow() : happiness_{7.5}
{
    // Nothing (else) to do.
}

Raptor::Raptor() : anger_{11.0}
{
    // Nothing (else) to do.
}
```

```
void Animal::speak() const
{
    cout << "??????" << endl;
}

void Cow::speak() const
{
    cout << "Mooooo" << endl;
}

void Raptor::speak() const
{
    cout << "Rawrrr" << endl;
}


void pet(Animal animal)
{
    animal.speak();
}


int main() {
    Cow bessie;
    Raptor peri;

    pet(bessie);
    pet(peri);

    return 0;
}
```