Announcements

- No programming over spring break
- Reading: Chapter 6
  - Start reading as soon as you have time
  - Quiz is already available
  - Quiz is due on the Monday you get back
    • Start of class.
- Expect PA6 to be handed out the Wednesday you get back
- Recitations are mandatory the week you return

Dispatch

- How methods are selected and invoked
- Methods/functions selected by signature
  - Name must match (e.g. WarmUp)
  - Arguments must match
    • Data types must match
    • Including hidden arguments
  - Return types not used for selection
- Polymorphism creates ambiguities
  - Animal::WarmUp();
  - Mammal::WarmUp();

Dispatch (II)

- Static Dispatch
  - Functions selected at compile time
  - Based on declared data types
  - This is the C++ default
- Dynamic Dispatch
  - Functions selected at run-time
  - Based on run-time data type of object
    • Run-time type is "sticky"
    • But does not apply to copies
    • Signalled by the 'virtual' keyword in C++

Under the Hood

- How is dynamic dispatch implemented?
- How is static dispatch implemented?
- How is inheritance implemented?

Back to Animals & Mammals...

class Animal {
public:
  bool warmup() {return false;}
  void Behave() {WarmUp();}
inline int& Lifespan() {return lifespan;}
protected:
  int lifespan;
};
class Mammal : public Animal {
public:
  bool warmup(){shiver(); return true;}
inline int& BodyTemp() {return body_temp;}
protected
  int body_temp;
};
Implementation: Animal

- How are Animal objects implemented?
  - What is `sizeof(Animal)`?
    - Caveat: implementations may differ...
    - Animal has 1 data field of type int
      `sizeof(Animal) == sizeof(int)` (potentially)
  - What about Animal's methods?
    - Implemented once per class
    - Not once per object
    - Executable code stored in memory

Static Dispatch

- None of Animal's methods are virtual
- Therefore all dispatch is static

```
// ignoring inline
Animal a;
a.LifeSpan() = 75;
```

Implementation: Mammal

- How are Mammal objects implemented?
  - What is `sizeof(Mammal)`?
    - Same caveat as before...
    - Every Mammal contains an Animal object
    - Mammals also have their own int field (1: body_temp)
      `sizeof(Mammal) == (sizeof(Animal) + sizeof(int))`
  - How is Mammal organized?
    - The first thing in a Mammal is its Animal
    - As a result, any Mammal* is also an Animal*

Organization of Inheritance

- Assume Quagga inherits Equine
- Equine inherits Mammal
- Mammal inherits Animal

Dynamic Dispatch

- If LifeSpan() is virtual, it function call can't be hardwired...

```
void Foo(Animal & a)
  a.LifeSpan() = 75;
```

- What happens?
The compiler doesn't know how 'a' was created
It might be an instance of a child class that overwrote LifeSpan[]...
So it doesn't know what function to call...

Back Again to Animals & Mammals...

class Animal {
public:
  bool warmup() {return false;}
  void Behave() {warmup();}
  virtual int & LifeSpan() {return lifespan;}
protected:
  int lifespan;
};
class Mammal : public Animal {
public:
  bool warmup() {shiver(); return true;}
  inline int & BodyTemp() {return body_temp;}
protected
  int body_temp;
};
Virtual Function Pointer Table (VFPT)

- Every class with virtual functions has a VFPT
- Every entry in this table is the address of a virtual method
- If LifeSpan() is the 1st entry for Animal, it is also the 1st entry for every derived class from Animal
- Note: one table per class, not per object

Implementing Dynamic Dispatch

- There is 1 exception to inheritance == union
- For objects with virtual functions
  - The first field is a pointer to the VFPT
  - Before any instances of parent classes
  - It points to the VFPT of the object as created

Implementing Dynamic Dispatch II

- Dynamic dispatch is implemented through the VFPT

```c
void Foo(Animal & a){
    a.LifeSpan() = 75;
}
```

What happens?
The compiler jumps to the address in the VFPT (1st field)
This will be the VFPT of the object as it was created
LifeSpan will be the 1st entry in this table (selected by name)
Jump to the code at that address