ECE 590.01  
C++ Programming, Data structures, and Algorithms 
Object Layout, Multiple Inheritance, and Mixins 

Admin 
- Reading 
  On class website 
- Homework 4 
  Due this Friday (April 5) 
- Project 
  Due April 12 
  In class: 10 min demos to everyone last week 
  Show off what you did 
  "Normal" demos w/ Tas: same week 

Talk this week: you should go! 
- Todd Austin is giving a talk next week 
  - Distinguished lecture series 
  - Thurs 3/4 @ 3:15 
  - Hudson 125 
- Attend if you can 

What have we been talking about? 
- What did we talk about last time? 
  - Algorithm Classes 
    - Greedy 
    - Dynamic Programming 
    - Brute Force 
    - Divide and Conquer 
  - Tries 

Now: some advanced topics in OOP 
- Advanced topics in OOP 
  - Multiple Inheritance 
    - Regular 
    - Virtual 
  - Object layout 
    - How does dynamic dispatch work? 
    - How do we make (multiple) inheritance work? 
  - Mixins 
    - Classes templated in terms of what their superclass is
Multiple Inheritance

- C++ supports multiple inheritance (Java does not):
  class Cyborg : public Human, public Robot {
     ...
  }

  Cyborg extends both Human and Robot
  - All the "normal" rules of inheritance apply
    - Polymorphism (can treat Cyborg as Human or Robot)
    - Inherits all fields/methods from both
    - Calls both parent constructors on creation

Multiple Inheritance (cont’d)

class Human {
public:
  void speak() { cout << "Good afternoon sir" << endl; }
};
class Robot {
public:
  void speak() { cout << "I am Bender, please insert girder" << endl; }
};
class Cyborg : public Human, public Robot;
int main(void) {
  Cyborg c;
  c.Human::speak();
  return 0;
}

Cyborg has two speak methods, need to say which one we mean

Multiple Inheritance (cont’d)

class Human {
public:
  virtual void speak() { cout << "Hello" << endl; }
};
class Robot {
public:
  void speak() { c.Human::speak();
      c.Cyborg::speak() default if you write c.speak()
        cout << "I am Bender, please insert girder" << endl;
    }
};
class Cyborg : public Human, public Robot{
public:
  void speak() {
    cout << "Resistance is futile." << endl;
    cout << "You will be assimilated" << endl;
  }
};

Note that if Cyborg overrides speak, there are now three
speak methods:

Human * h = new Cyborg();  h->speak();

Multiple Inheritance (cont’d)

class Human {
public:
  virtual void speak() { cout << "Hello" << endl; }
};
class Robot {
public:
  virtual void speak() { cout << "I am Bender, please insert girder" << endl; }
};
class Cyborg : public Human, public Robot {
public:
  virtual void speak() { cout << "RIF. YWBA." << endl; }
};

Human * h = new Cyborg();  h->speak();

What does this print?
Multiple Inheritance (cont’d)

```cpp
class Human {
public:
    virtual void speak() { cout << "Hello" << endl; }
};

class Robot {
public:
    void speak() { cout << "I am Bender, please insert girder" << endl; }
};
class Cyborg : public Human, public Robot{
public:
    void speak() { cout << "RIF. YWBA." << endl; }
};

Human * h = new Cyborg();
h->speak();
```

ECE 590.01 (Hilton): Advanced OOP

13

Now what happens?

```cpp
Robot * r = new Cyborg();
r->speak();
```

ECE 590.01 (Hilton): Advanced OOP

14

More Multiple Inheritance Issues

```cpp
class GuiObject {
protected:
    int x, y;
};
class Button : public GuiObject {};
class ImgDisplay : public GuiObject {};
class ImgButton : public Button, public ImgDisplay {};

ImgButton inherits from two GuiObjects: Button and ImgDisplay
GuiObject has x,y coordinates...
How many x's and y's does ImgButton have?
```

ECE 590.01 (Hilton): Advanced OOP

16

Drawing out the inheritance hierarchy show this
Two separate (distinct) GuiObject parents

In this case: bad for our design. Resulting object is one
GuiObject: has one (x,y) location

Asking for trouble
More Multiple Inheritance Issues

```cpp
class GuiObject {
protected:
    int x, y;
};
class Button : public GuiObject {
};
class ImgDisplay : public GuiObject {
};
class ImgButton : public Button, public ImgDisplay {
}
```

Can we make this work with ONE GuiObject? (as shown above)

Virtual Inheritance

```cpp
class GuiObject {
protected:
    int x, y;
};
class Button : public virtual GuiObject {
};
class ImgDisplay : public virtual GuiObject {
};
class ImgButton : public Button, public ImgDisplay {
}
```

Can we make this work with ONE GuiObject? (as shown above)

Yes: virtual inheritance
Specify at Button and ImgDisplay (need on both)
Says "Only one GuiObject if its multiply inherited by children"

How does all of this work?

- All of these high level OOP constructs sound great
  - Classes
  - Inheritance
  - Polymorphism
  - Dynamic Dispatch
  - Multiple Inheritance
- But they all must be implemented
  - Words in memory
  - Assembly instructions
- Next up: we learn how
  - Good to know in general
  - Understand performance cost associated with these features

Start simple

- No inheritance, no virtual methods
```cpp
class A {
public:
    int x;
    int y;
    int getXplusY() { return x + y; }
};
```

Note: multiple classes may have same method name. compiler "mangles" name to encode class name + method name.

Start simple

- Fields (x,y) live in object
- One copy of code (outside object), takes "this" param
```cpp
struct A {
    x  
    int x;
    int y;
};
```

```
g getXplusY(A* this) {
    return this->x + this->y;
}
```

Note: multiple classes may have same method name. compiler "mangles" name to encode class name + method name.
Inheritance: No virtual functions (yet)

```cpp
class A{
  int x, y;
  int getXplusY() { return x + y; }
};
class B : public A {
  int z;
  int XYZ() { return getXplusY() + z; }
};

Need to support polymorphism:
A * a = new B();
```

Virtual Functions

```cpp
class A{
  int x, y;
  virtual int getSum() { return x + y; }
};
class B : public A {
  int z;
  virtual int getSum() { return x + y + z; }
  virtual int getProd() { return x * y * z; }
};

A * a = new B();
int s = a->getSum(); // how to figure out what to call?
// (note: compiler cannot determine)
```

Dynamic dispatch

- How do we solve all functionality problems?

Add a level of indirection

Dynamic dispatch

- How do we solve all functionality problems?

Virtual Functions

```cpp
class A{
  int x, y;
  virtual int getSum() { return x + y; }
};
class B : public A {
  int z;
  virtual int getSum() { return x + y + z; }
  virtual int getProd() { return x * y * z; }
};

A * a = new B();
int s = a->getSum(); // how to figure out what to call?
// (note: compiler cannot determine)
```

Dynamic dispatch

- How do we solve all functionality problems?

Add a level of indirection

VTable

- Objects with virtual functions have a pointer to a "vtable"
  - "Virtual function table"
  - Holds pointers to functions
  - This is why you need virtual in the parent, not the child

- Dynamic dispatch:
  - Read vtable pointer from object (fixed slot, typically 0)
  - Read function pointer from vtable
  - Vtable must respect "sub-objecting" design
  - Call the function via the pointer read from the table (jalr)

- Performance Cost (vs static dispatch): two loads
  - May also prevent optimizations (e.g., inlining)
  - Could be avoided in some cases (e.g., apply CFA [Shivers, 88])
class A {
    int x, y;
    virtual int getSum() {
        return x + y;
    }
}

class B : public A {
    int z;
    virtual int getSum() {
        return x + y + z;
    }
    virtual int getProd() {
        return x * y * z;
    }
};

ECE 590.01 (Hilton): Advanced OOP

Abstract functions

class A {
    int x, y;
    virtual int getSum() {
        return x + y;
    }
    virtual void foo() = 0;
};

Abstract function: Vtable entry is NULL
Should never be called (if we manage to, program segfaults)

Multiple Inheritance Layout

class A {
    int x;
    virtual void f() { ... }
}

class B {
    int y;
    virtual void g() { ... }
}

class C: public A, public B {
    int z;
    virtual void f() { ... }
    virtual void g() { ... }
}
Multiple Inheritance Layout

class A{
    int x;
    virtual void f() {} ...
};
class B{
    int y;
    virtual void g() {} ...
};
class C: public A, public B {
    int z;
    virtual void f() {} ...
    virtual void g() {} ...
};

ECE 590.01 (Hilton): Advanced OOP

37

Naïve approach does not work

Why not?

No B sub-object!

Now: no A sub-object

Solution:

Have sub-object, not at start

Cast (implicit or explicit) to non-primary parent class type
results in different pointer

Cost: add immediate instruction
Multiple Inheritance Layout

```
C * c = new C();
B* b = c;
b->g();
```

But wait…

What g? Does this dispatch to?

```
class C: public A, public B {
  int z;
  virtual void f() {…}
  virtual void g() {…}
};
```

Dynamic dispatch to C's g()

What type does it expect for "this"?

```
this = this - 8
```

• Adjustment to "this" before function
• C's B sub-object vtable points at $v_{g}$
• C's regular vtable points at "normal code"

Earlier example re-visited

```
class GuiObject {
  int x, y;
  virtual void f() {…}
};
class Button : public GuiObject {
  int z;
  virtual void g() {…}
};
class ImgDisplay: public GuiObject {
  int q;
  virtual void h() {…}
};
class ImgButton: public Button, public ImgDisplay {
  int k;
};
```

Try to lay this out
(Not virtual inheritance—yet)
Earlier example re-visited

```cpp
class GuiObject {
    int x, y;
    virtual void f() {...}
};
class Button : public GuiObject {
    int z;
    virtual void g() {...}
};
class ImgDisplay : public GuiObject {
    int q;
    virtual void h() {...}
};
class ImgButton : public Button, public ImgDisplay {
    int k;
}
```

Virtual Multiple Inheritance

```cpp
class GuiObject {
    int x, y;
    virtual void f() {...}
};
class Button : public virtual GuiObject {
    int z;
    virtual void g() {...}
};
class ImgDisplay : public virtual GuiObject {
    int q;
    virtual void h() {...}
};
class ImgButton : public Button, public ImgDisplay {
    int k;
}
```

Now make it virtual

```cpp
class GuiObject {
    int x, y;
    virtual void f() {...}
};
class Button : public virtual GuiObject {
    int z;
    virtual void g() {...}
};
class ImgDisplay : public virtual GuiObject {
    int q;
    virtual void h() {...}
};
class ImgButton : public Button, public ImgDisplay {
    int k;
}
```

Start with just this part
Virtual Multiple Inheritance

class GuiObject {
    int x, y;
    virtual void f() {...}
};
class Button : public virtual GuiObject {
    int z;
    virtual void g() {...}
};

Button * b = ...
GuiObject * g = b;

ECE 590.01 (Hilton): Advanced OOP

Now make it virtual

GuiObject is same

Virtual Multiple Inheritance

class GuiObject {
    int x, y;
    virtual void f() {...}
};
class Button : public virtual GuiObject {
    int z;
    virtual void g() {...}
};

Now make it virtual

Button has virtual inheritance

Vtable has offset to parent

Virtual Multiple Inheritance

class GuiObject {
    int x, y;
    virtual void f() {...}
};
class Button : public virtual GuiObject {
    int z;
    virtual void g() {...}
};

Button * b = ...
GuiObject * g = b;
g->f();

Dynamically dispatched calls may require fixup

f's implementation may be in a subclass (e.g. B)
GuiObject's vtable points to fixup before
"normal" codes

Virtual Multiple Inheritance

class GuiObject {
    int x, y;
    virtual void f() {...}
};
class Button : public virtual GuiObject {
    int z;
    virtual void g() {...}
};

Other parent class (ImgDisplay)
is similar

ImgDisplay: public virtual GuiObject {
    int q;
    virtual void h() {...}
};

Vtable> <parent> +8
v1 vb> f
v2 vb> g
v3 vb> h
v4 vb> k

Virtual Multiple Inheritance

class GuiObject {
    int x, y;
    virtual void f() {...}
};
class Button : public virtual GuiObject {
    int z;
    virtual void g() {...}
};
class ImgDisplay: public virtual GuiObject {
    int q;
    virtual void h() {...}
};
class ImgButton: public Button, public ImgDisplay {
    int k;

ECE 590.01 (Hilton): Advanced OOP
Virtual Multiple Inheritance

ImgButton layout
Level of indirection allows "holes"

Virtual Multiple Inheritance

ImgButton layout
Level of indirection allows "holes"

Cast as a Button
GuiObject access: look in vtable

Access to x?
Load vtbl pointer
Load parent_offset
Compute parent = parent_offset + this
Load parent_offset + field offset

Constrain with "Normal": just (this + field offset)

Multiple Inheritance Summary

- Multiple Inheritance
  - Two options: regular + virtual
  - Performance costs
    - Understand them if you want to use it
    - Are they worth it?
    - Designer time usually more important than run time...
- Java does not have MI
  - Avoids the entire can of worms
  - Does have interfaces
    - No fields, only methods (avoids field layout issues)
    - Still need to solve vtable layout issues

Mixins

- Imagine working on a GUI
- Writing a class that puts fancy borders around any other component:
  - Would like to extend the component we are putting borders around
  - Effectively be a new component just like it, but with fancy borders
- Use a mixin: template over parent class

Mixins

- More generally:
  - Add functionality to something with a given interface
  - ...typically keeping the same interface
  - Can be layered together as needed
- Read Yannis Smaragdakis's paper for more

Wrap up for today

- Multiple Inheritance
- Object layout
  - Inheritance
  - Polymorphism
  - Dynamic Dispatch
  - Multiple Inheritance
  - Mixins
- Next time:
  - A bit about garbage collection