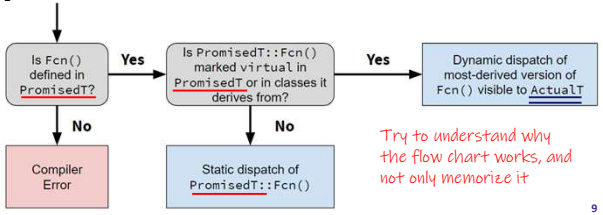
Dispatch Decision Tree

- Which function is called is a mix of both compile time and runtime decisions as well as *how* you call the function
  - If called on an object (*e.g.* `obj.Fcn()`), usually optimized into a hard-coded function call at compile time
  - If called via a pointer or reference:

```
PromisedT* ptr = new ActualT;
ptr->Fcn();  // which version is called?
```



Try to understand why the flow chart works, and not only memorize it

Poll Everywhere

pollev.com/cse33320su

- Apply what you've learned to a more complex example!
- What is printed?

poll.cc

```
class A {
 public:
  virtual void Foo() {
    cout << "H";
    this->Bar();
  }

  void Bar() {
    cout << "A";
  }
};

class B : public A {
 public:
  virtual void Bar() {
    cout << "I";
  }
};
```

```
int main() {
  B b;
  B* b_ptr = &b;

  // Q:
  b_ptr->Foo();
}
```

A. HI
B. HA
C. Compiler Error
D. Segmentation fault
E. We're lost…

Abstract Classes

- Sometimes we want to include a function in a class but *only* implement it in derived classes
  - In Java, we would use an abstract method
  - In C++, we use a "pure virtual" function
    - Example: `virtual string noise() = 0;`
- A class containing *any* pure virtual methods is abstract
  - You can't create instances of an abstract class
  - Extend abstract classes and override methods to use them
- A class containing *only* pure virtual methods is the same as a Java interface
  - Pure type specification without implementations

Casting in C++

- C++ provides an alternative casting style that is more informative:
  - `static_cast<to_type>(expression)`
  - `dynamic_cast<to_type>(expression)`
  - `const_cast<to_type>(expression)`
  - `reinterpret_cast<to_type>(expression)`
- Always use these in C++ code
  - Intent is clearer
  - Easier to find in code via searching