13. *Polymorphism: Usage Mechanism*

The Effect of the Virtual keyword

- virtual specifier on a function indicates to a caller that an object of a derived class may use an alternate implementation

- If a function is not virtual, it is always statically bound - on references, pointers and objects

- If a function is virtual, it is dynamically bound on references and pointers, not on objects
Example of Binding

class X {
public:
    virtual void f1();
    void f2();
    virtual void f3()
    {
        f1();
        f2();
    }
};

X obj;
obj.f1();
obj.f2();
obj.f3();

X* ptr = …
ptr->f1();
ptr->f2();
ptr->f3();

X& ref = …
ref.f1();
ref.f2();
ref.f3();

Virtual Tables & vptr

Efficient Implementation of Polymorphism

Example:
class Vehicle {...
    virtual void drive();
    virtual void FillGas();
};

class Car : public Vehicle {...
    virtual void drive();
    virtual void RollWindow(int);
};

class MotorBike :
    public Vehicle {...
        virtual void drive();
    };

Vehicle v1;
Car c1, c2;
MotorBike m1;
The VTable for the Vehicle Example

- MotorBike’s Vtable
  - Vehicle::drive() {...}
  - Vehicle::FillGas() {...}
  - Car::drive() {...}
  - Car::RollWind(int) {...}
- Car’s vtable
  - Vehicle::drive() {...}
- Vehicle’s Vtable
  - Vehicle::drive() {...}

Virtual Destructor

```cpp
Employee* eptr1 = new Employee;
Employee* eptr2 = new Manager;
fn(eptr1);
fn(eptr2);
```

```cpp
void fn(Employee* eptr)
{
  ...
  delete eptr; // Should call ~Employee() or
  ~Manager() ???
}
```

- "If a Class has a virtual function, it needs a virtual destructor"
- If you expect class to be used as Base class, you must write a virtual destructor
!? Virtual Constructors ?!

- Not available - no virtual constructors in C++
- But what's the Need?

Example:

class Car {
    ...
    Engine* engineptr;
    Car& operator=(const Car& obj) {
        ...
        engineptr = new Engine;
        *engineptr = *(obj.engineptr);
    }
};

Car c1, c2;
c1.SetEngine(aTurboEngine);
c2 = c1;

Solution: Prototype Pattern

Car Car::operator(const Car& obj) {
    ...
    engineptr = obj.engineptr->createnew();
    *engineptr = *(obj.engineptr);
}

class Engine {...
    virtual Engine* createnew() {
        return new Engine;
    }
};

class TurboEngine : Engine {
    virtual Engine* createnew() {
        return new TurboEngine;
    }
};
Pure Virtual Functions - Revisited

A Pure Virtual Function is a Virtual Function with no implementation

class Employee { ... 
    virtual void YearlyReview() = 0; // Pure Virtual Function 
};

• Class with Pure Virtual Function is an Abstract Base class
• No instances of an ABC may be created
• An ABC is used to represent Abstract concepts and to inherit other classes from
• A class derived from an ABC must provide an implementation for the pure virtual function to be a concrete (i.e., a non ABC) class

Caution: Calling Virtual Functions in Constructors & Destructors

class Employee { ...
    virtual void print() { cout << “name”; } 
    Employee(const char* name)
    { 
        cout << “Creating Employee: “;
        print(); // Which print called here?
    }
    virtual void report()
    { 
        print(); // Which print called here?
    }
};

class Manager : public Employee { ...
    virtual void print()
    { 
        Employee::print();
        cout << “level”;
    }
    Manager (const char* name, int level)
    : Employee(name)
    { 
        cout << “Creating Manager: “;
        print(); 
    }
};
Lab Work: Details provided on-line.