Today

- Uses of Inheritance in C++
- Abstract Classes
- Run-Time Identification

Multiple Inheritance

```
class A {
public:
  A () { cout << " create an object of type A\n"; }
  ~A () { cout << " destroy an object of type A\n"; }
};

class B {
public:
  B () { cout << " create an object of type B\n"; }
  ~B () { cout << " destroy an object of type B\n"; }
};

class C : public A, public B {
public:
  C () : A(), B() { cout << " create an object of type C\n"; }
  ~C () { cout << " destroy an object of type C\n"; }
};
```

Virtual Inheritance

```
class A : public O {
public:
  A () { cout << " create an object of type A\n"; }
  ~A () { cout << " destroy an object of type A\n"; }
};

class B : public O {
public:
  B () { cout << " create an object of type B\n"; }
  ~B () { cout << " destroy an object of type B\n"; }
};

class C : public A, public B {
public:
  C () : A(), B(), O () { cout << " create an object of type C\n"; }
  ~C () { cout << " destroy an object of type C\n"; }
};
```

Virtual Base Class Member

```
class A : virtual public O {
public:
  A () { cout << " create an object of type A\n"; }
  ~A () { cout << " destroy an object of type A\n"; }
};

class B : virtual public O {
public:
  B () { cout << " create an object of type B\n"; }
  ~B () { cout << " destroy an object of type B\n"; }
};

class C : public A, public B {
public:
  C () : A(), B() { cout << " create an object of type C\n"; }
  ~C () { cout << " destroy an object of type C\n"; }
};
```

Public, Private& Protected

- public -> type inheritance
- private -> implementation inheritance
- protected -> ?

Inheritance & Scopes

```
class A {
public:
  void print () { cout << " print from A" << endl; }
};

class B : public A { };

class C {
public:
  void print () { cout << " print from C" << endl; }
};

class D : public C, public B {
  void print () { print (); } // compilation error ???
  void print2 () { A::print (); } // or B::print ();
};
```

Polymorphism

- Goal: one interface, multiple methods
- Advantage: easier to memorate such a program

Polymorphism

```
// Common interface

class A {
public:
  A *clone () = 0;

  virtual void store (int) = 0;
  virtual void retrieve (int&) = 0;

  virtual void print () = 0;
  virtual void print (ostream&) = 0;
  virtual void read () = 0;
  virtual void read (istream&) = 0;
};
```

Run-Time Identification

- **typeid and type_info class**

- **Example:**

```
# include <typeinfo>
cout << typeid (*a).name ()
     << typeid (Line).name () << endl;

class type_info {
  // Implementation dependent
private:
  type_info (const type_info&);
  type_info& operator = (const type_info&);
public:
  virtual ~type_info ();
  int operator== (const type_info&) const;
  int operator!= (const type_info&) const;
  const char* name () const;
};
```