Run-Time Type Identification

Jim Fawcett
CSE687 – Object Oriented Design
Spring 2007
**typeid**

- The function:
  ```cpp
cost typeinfo typeid(arg)
  ```
  returns a const typeinfo object when passed an object, reference, or pointer.

- Typeinfo:
  ```cpp
class type_info {
  public:
    virtual ~type_info();
    int operator==(const type_info& rhs) const;
    int operator!=(const type_info& rhs) const;
    int before(const type_info& rhs) const;
    const char* name() const;
    const char* raw_name() const;
  private:
    ...
  }
  ```
Examples of typeid() use

- class foo { ... };
  typeid.foo().name() returns “foo”

- foo *ptr;
  typeid(ptr).name() returns “foo*”
  typeid(*ptr).name() returns “foo”

- class derived : public base { ... };
  typeid.base().before(typeid(derived)) returns true
Shape *sp = new circle;

typeid(shape) == typeid(*sp) returns false
typeid(shape).before(typeid(*sp)) returns true
typeid(sp).name() returns "circle*
typeid(*sp).name() returns "circle"
**dynamic_cast<...>(...)**

- Dynamic casts support safe down-casting (casting down an inheritance hierarchy):

```cpp
Line* lptr = new line;
circle* cptr = new circle;
rectangle* rptr = new rectangle;
shape* array[3] = { lptr; cptr, rptr }; 
for(int i=0; i<3; i++)
    if(dynamic_cast<circle*>(array[i]))
        cout << “circle” << endl;
    else
        cout << “non-circle” << endl;
```
Caution

- Typeid and dynamic_cast information is carried in a class’s virtual function pointer table and is intended to be used only with polymorphic classes, e.g., those with at least one virtual function.

- Typeid works for non-polymorphic classes, but returns only static type info, e.g., based on the static pointer type, not on the type of the object pointed to.

- You must enable run-time type information (RTTI) in your project settings (C/C++ tab, C++ language category). Your program will crash if you use dynamic_cast or RTTI and forget to do this.
Enabling RTTI

On by default
End of Presentation