CS2810 OOAIA

Runtime Type Information

Example Code Snippets

1. Basic Type Checking with `typeid`

```
#include <typeinfo>
#include <iostream>

class Shape {
public:
    virtual ~Shape() {} // Polymorphic base
};

class Circle : public Shape {};
class Square : public Shape {};

int main() {
    Shape* shape = new Circle();

    std::cout << "Type: " << typeid(*shape).name() << "\n"; //
Output: 6Circle

    if(typeid(*shape) == typeid(Circle)) {
        std::cout << "Circle detected!\n";
    }

    delete shape;
}
```

```
// typeid works on object instances, returns mangled type name
```

2. Safe Downcasting with `dynamic_cast`

```
class Animal {
public:
    virtual ~Animal() {}
    virtual void speak() = 0;
};

class Dog : public Animal {
public:
    void speak() override { std::cout << "Woof!\n"; }
    void fetch() { std::cout << "Fetching...\n"; }
};

void interact(Animal* animal) {
    if(Dog* dog = dynamic_cast<Dog*>(animal)) {
        dog->fetch(); // Safe to call Dog-specific method
    }
}

int main() {
    Animal* pet = new Dog();
    interact(pet); // Output: Fetching...
    delete pet;
}

// Safe access to derived class features
```

3. Handling Failed Casts

```
class Base { virtual ~Base() {} };
class Derived : public Base {};

int main() {
    Base* b1 = new Base();
```

```
    Base* b2 = new Derived();

    Derived* d1 = dynamic_cast<Derived*>(b1); // Returns nullptr
    Derived* d2 = dynamic_cast<Derived*>(b2); // Succeeds

    std::cout << "d1: " << (d1 ? "Valid" : "Invalid") << "\n"; //
Invalid
    std::cout << "d2: " << (d2 ? "Valid" : "Invalid") << "\n"; //
Valid
}

// dynamic_cast returns nullptr for invalid pointer casts
```

4. RTTI with Multiple Inheritance

```
class Flying {
public:
    virtual ~Flying() {}
    virtual void fly() = 0;
};

class Bird : public Animal, public Flying {
public:
    void speak() override { std::cout << "Chirp!\n"; }
    void fly() override { std::cout << "Flying!\n"; }
};

int main() {
    Animal* a = new Bird();

    if(Flying* f = dynamic_cast<Flying*>(a)) {
        f->fly(); // Cross-cast works with RTTI
    }
}

// Cross-casting in diamond hierarchies
```

5. Type Comparison with References

```
void processShape(Shape& s) {
    try {
        Circle& c = dynamic_cast<Circle&>(s);
        std::cout << "Processing circle\n";
    } catch(const std::bad_cast& e) {
        std::cout << "Not a circle: " << e.what() << "\n";
    }
}

int main() {
    Square sq;
    processShape(sq); // Output: Not a circle: std::bad_cast
}

// dynamic_cast with references throws bad_cast
```

6. Custom Type Information

```
#include <typeinfo>
#include <map>

class CustomRTTI {
    static std::map<const std::type_info*, std::string> typeNames;
public:
    template<typename T>
    static void RegisterType(const std::string& name) {
        typeNames[&typeid(T)] = name;
    }

    template<typename T>
    static std::string GetTypeName() {
        return typeNames[&typeid(T)];
    }
};

// Initialize static member
std::map<const std::type_info*, std::string> CustomRTTI::typeNames;
```

```
int main() {
    CustomRTTI::RegisterType<Circle>("AwesomeCircle");
    std::cout << CustomRTTI::GetTypeName<Circle>(); // Output:
AwesomeCircle
}

// Extending RTTI with custom metadata
```

Practice Problems

1. Problem: Write a function `is_dynamic_castable(Base* ptr)` that::

- Returns `true` if `ptr` can be safely downcast to `Derived*` using `dynamic_cast`
- Works with any classes inheriting from `Base`
- Returns `false` for null pointers

Demonstrate proper use of `dynamic_cast` and template type deduction.

Example:

```
class Base { virtual ~Base() {} };
class Derived : public Base {};

Base* b1 = new Derived;
Base* b2 = new Base;

std::cout << is_dynamic_castable<Derived>(b1); // 1 (true)
std::cout << is_dynamic_castable<Derived>(b2); // 0 (false)
```

2. Problem: Create a function `are_same_type(const auto& a, const auto& b)` that:

- Returns `true` if both arguments have the same dynamic type
- Works correctly with polymorphic objects
- Uses RTTI mechanisms

Highlight difference between static and dynamic type comparison.

Example:

```
Animal* a = new Dog;
Animal* b = new Cat;
std::cout << are_same_type(*a, *b); // 0
```

3. Problem: Implement a function `safe_reference_cast(Base& ref)` that:

- Attempts to cast reference to `Derived&` using `dynamic_cast`
- Returns `true` if cast succeeds, `false` if `bad_cast` occurs
- Doesn't use exception handling in caller code

Example:

```
try {
    Base& b = get_object();
    if(safe_reference_cast<Derived>(b)) {
        // Use derived features
    }
} catch(...) { /* Handle other errors */ }
```

4. Problem: Create a function `demangle_type_name(const auto& obj)` that:

- Returns demangled type name using `typeid`
- Works with both fundamental and user-defined types
- Uses compiler-specific demangling (e.g., `abi::__cxa_demangle`)

Example:

```
std::vector<int> v;
std::cout << demangle_type_name(v); // "std::vector<int,
std::allocator<int>>"
```

5. Problem: Implement a template function `check_polymorphic<T>()` that:

- Fails to compile if `T` isn't polymorphic
- Uses RTTI features to enforce this
- Provides clear error messages

Example:

```
check_polymorphic<Base>(); // OK (virtual dtor)
check_polymorphic<int>();  // Compile error
```