Class A*

Caught Attempted a typeid of Null pointer!

* we should not use typeid as a practice as it uses RTTI.

When should we use const_cast in C++?

(1) passing const data to a function that does not receive const.

```
int func(int *ptr)        // legacy code
{
    return (*ptr);
}

main
{
    const int val = 5;
    const int *ptr = &val;
    int *ptr1 = const_cast<int*>(ptr);
    fun(ptr1);
}
```

(2) change non-const class member inside a const member fn.

```
class A
{
    int n;
public:
    void f(int i) const {        // Here this pointer is const so we need to cast away its constness.
        // this->n = i;          // Not allowed.
        const_cast<A*>(this)->n = i;
    }
```

241-1

Note: except that any class member declared mutable can be modified any attempt to modify a const object during its lifetime results in undefined behaviour

Note: reinterpret_cast cannot cast away the const, volatile or --unaligned attribu.

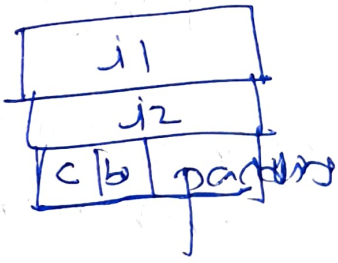
```
int a;
const int *ap = &a;
char *pc = reinterpret_cast<char*>(ap);
```

fail ✗

```
const char *pc = reinterpret_cast<const
    char*>(ap) ✓
```

(11) It can be used to bit manipulation in struct.

```
struct s
{   int i1
    int i2
    char c
    bool
};
```




```
struct s st;
int *ps = reinterpret_cast<int*>(&st);
ps++; for i2
```

241.2

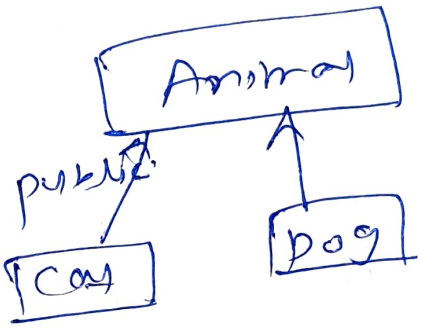
RTTI (Runtime type identification)

* provides a standard way for a pgm to determine the type of object during runtime provided through.

1. typeid: return the actual type of object referred to by pointer or reference

2. dynamic_cast: safely convert from pointer or reference of base type to derived type

ex.



⇒ Here base class has to be polymorphic

```
Animal *bp1 = new Cat();
"      *bp2 = "   Dog();

Dog* pD = dynamic_cast<Dog*>(bp1);
```

return null since bp1 is not of type Cat

```
dynamic_cast<Dog*>(bp2)
```

return pointer

If base class is not polymorphic then it will be compiler error.

explicit keyword in c++ (241-3)

→ avoid implicit calls to constructor or implicit constructor conversion