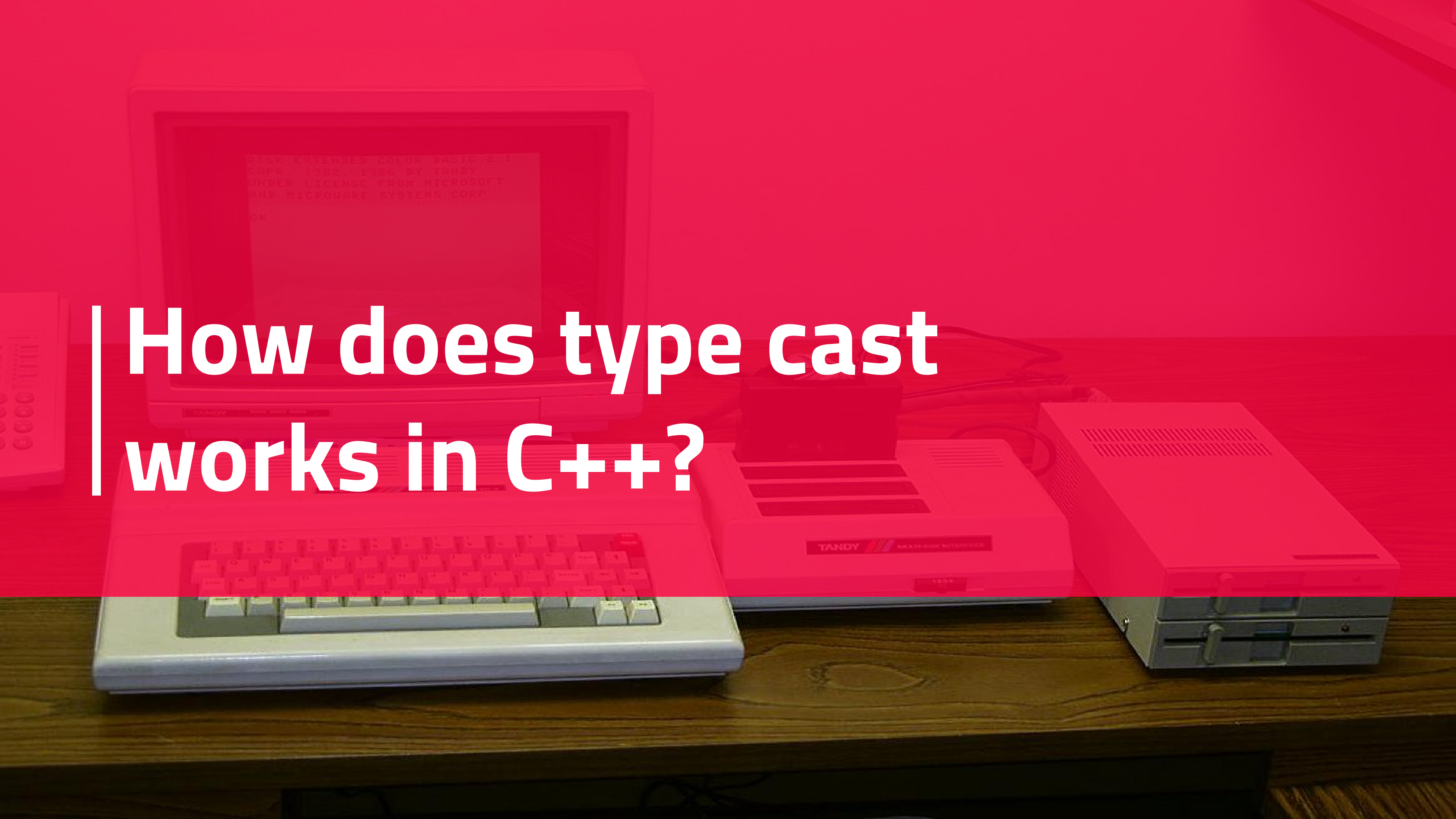
How does type cast works in C++?



DISK EXTENDED COLOR BASIC 2.1
COPR. 1982, 1986 BY TANDY
UNDER LICENSE FROM MICROSOFT
AND MICROWARE SYSTEMS CORP.
OK
TANDY

Hello!

Higor Anjos

11403767

Marcos Alves

11409511

1.
What is a cast?

> *A cast is a mechanism that converts a value from one data type to another data type.*
>
> *-IBM Knowledge Center*

2.
Types of cast in C++?

There are 4 types
Of casts

1. static_cast
2. dynamic_cast
3. reinterpret_cast
4. const_cast

STATIC CAST

STATIC CAST

Substitute for C notation

Code sample

```
int main(){

    Triangle *triangle = new Triangle;
    Shape *shapec = (Shape *) triangle;









    return 0;
}
```

Code sample

```
int main(){

        Triangle *triangle = new Triangle;
        Shape *shapec = (Shape *) triangle;
        Shape *shapecpp = static_cast<Shape*>(triangle);

        return 0;
}
```

Code sample

```
int main(){

        Triangle *triangle = new Triangle;
        Shape *shapec = (Shape *) triangle;
        Shape *shapecpp = static_cast<Shape*>(triangle);

        cout << “Shape C vertices: ”;
        cout << shapec->num_vertice << endl;
        cout << “Shape Cpp vertices: ”;
        cout << shapecpp->num_vertice << endl;



        return 0;
}
```

Code sample

```
int main(){

        Triangle *triangle = new Triangle;
        Shape *shapec = (Shape *) triangle;
        Shape *shapecpp = static_cast<Shape*>(triangle);

        cout << “Shape C vertices: ”;
        cout << shapec->num_vertice << endl;
        cout << “Shape Cpp vertices: ”;
        cout << shapecpp->num_vertice << endl;



        return 0;
}
```

Output

```
Shape C vertices: 3
Shape Cpp vertices: 3
```

STATIC_CAST

Substitute for C notation.

STATIC CAST

Substitute for C notation. Always?

Code sample

```
int main(){

        Microwave *micro = new Microwave;
        Soap *soapc = (Soap *) micro;



        return 0;
}
```

Code sample

```
int main(){

    Microwave *micro = new Microwave;
    Soap *soapc = (Soap *) micro;


    cout << "Soap C:" << endl;
    cout << soapc->type << endl;




    return 0;
}
```

Code sample

```
int main(){

        Microwave *micro = new Microwave;
        Soap *soapc = (Soap *) micro;


        cout << "Soap C:" << endl;
        cout << soapc->type << endl;




        return 0;
}
```

Output

```
Soap C:
I’m a microwave
```

Code sample

```
int main(){

        Microwave *micro = new Microwave;
        Soap *soapc = (Soap *) micro;
        Soap *soapcpp = static_cast<Soap*>(micro);

        cout << "Soap C:" << endl;
        cout << soapc->type << endl;




        return 0;
}
```

Output

```
Soap C:
I’m a microwave
```

Code sample

```
int main(){

        Microwave *micro = new Microwave;
        Soap *soapc = (Soap *) micro;
        Soap *soapcpp = static_cast<Soap*>(micro);

        cout << "Soap C:" << endl;
        cout << soapc->type << endl;




        return 0;
}
```

Output

```
error: invalid static_cast
from type ‘Microwave*’
to type ‘Soap*’
```

Code sample

```
int main(){

    Microwave *micro = new Microwave;
    Soap *soapc = (Soap *) micro;
    Soap *soapcpp = static_cast<Soap*>(micro);

    cout << "Soap C:" << endl;
    cout << soapc->type << endl;




    return 0;
}
```

Output

```
error: invalid static_cast
from type ‘Microwave*’
to type ‘Soap*’
```

STATIC CAST

Function Overload and Templates

Code sample

```
template <typename T>
T foo(T n){
        return n/2;
}
```

Code sample

```
template <typename T>
T foo(T n){
        return n/2;
}

int main(){

        cout << foo(7) << endl;



        return 0;
}
```

Code sample

```
template <typename T>
T foo(T n){
        return n/2;
}

int main(){

        cout << foo(7) << endl;



        return 0;
}
```

Output

```
3
```

Code sample

```
template <typename T>
T foo(T n){
	return n/2;
}

int main(){

	cout << foo(7) << endl;
	cout << foo(static_cast<double>(7)) << endl;


	return 0;
}
```

Output

```
3
```

Code sample

```
template <typename T>
T foo(T n){
        return n/2;
}

int main(){

        cout << foo(7) << endl;
        cout << foo(static_cast<double>(7)) << endl;


        return 0;
}
```

Output

```
3
3.5
```

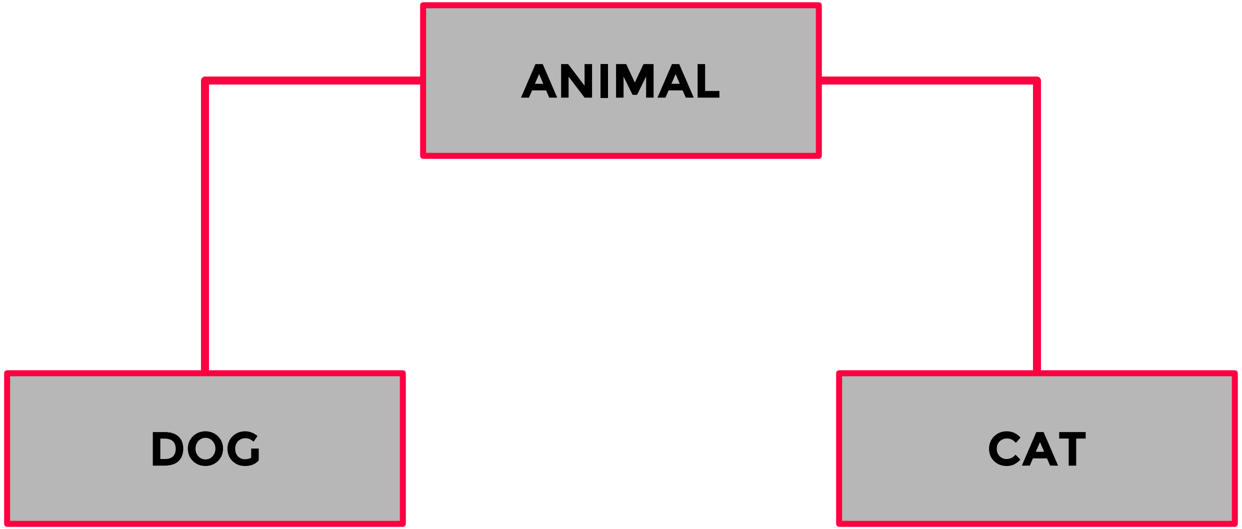
STATIC CAST

No run-time checking

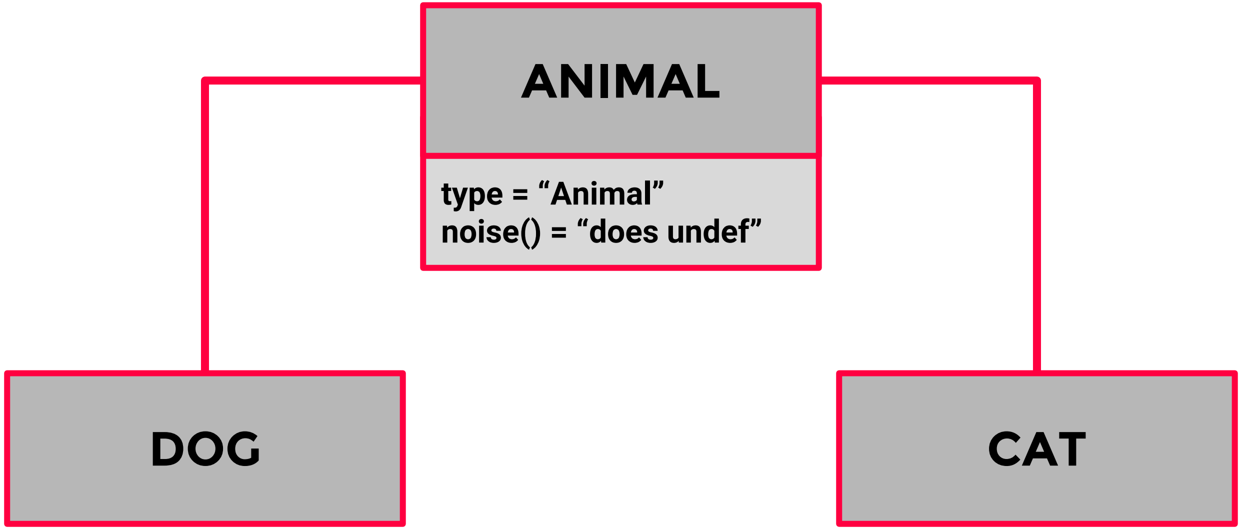
Let’s create a scenario

ANIMAL

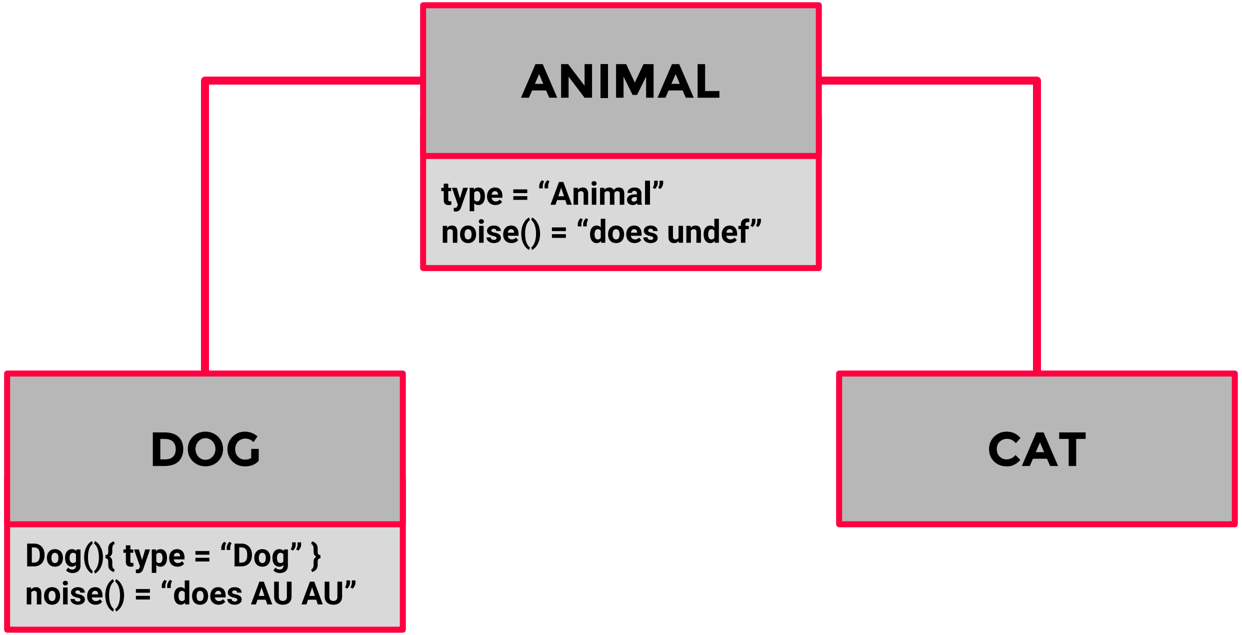
Let’s create a scenario



Let’s create a scenario



Let’s create a scenario



ANIMAL

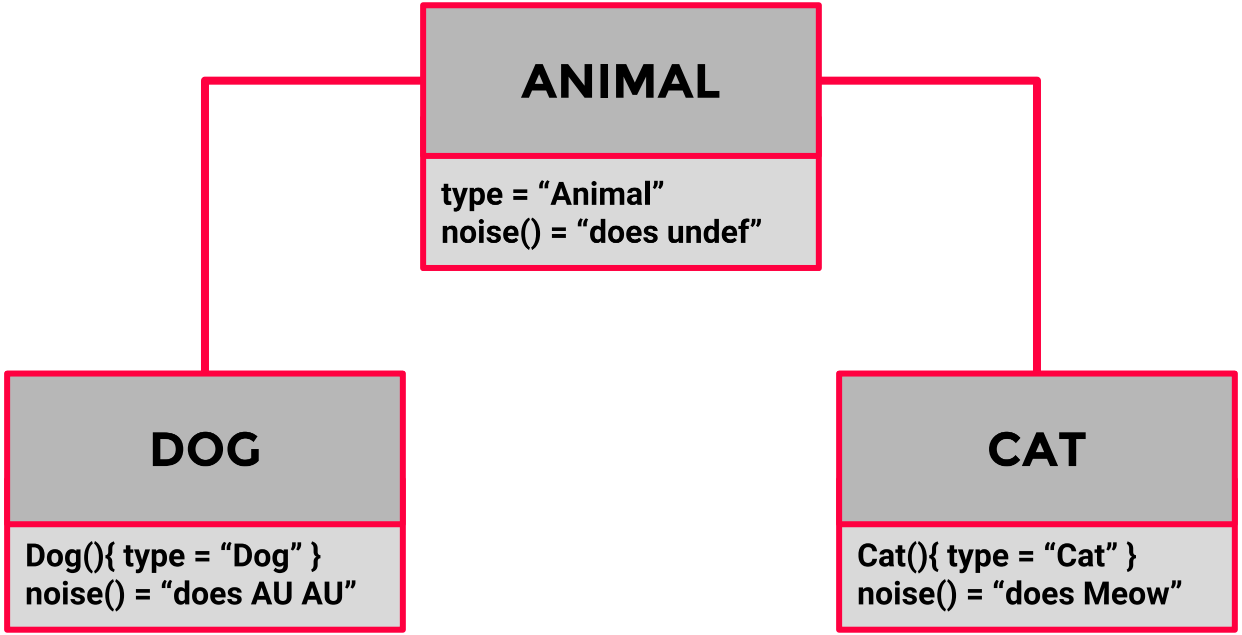
type = “Animal”
noise() = “does undef”

DOG

Dog(){ type = “Dog” }
noise() = “does AU AU”

CAT

Let’s create a scenario



ANIMAL

type = “Animal”
noise() = “does undef”

DOG

Dog(){ type = “Dog” }
noise() = “does AU AU”

CAT

Cat(){ type = “Cat” }
noise() = “does Meow”

Code sample

```
int main(){

        Animal* animal = new Dog;







        return 0;
}
```

Code sample

```
int main(){

        Animal* animal = new Dog;
        Cat* cat = static_cast<Cat*>(animal);





        return 0;
}
```

Code sample

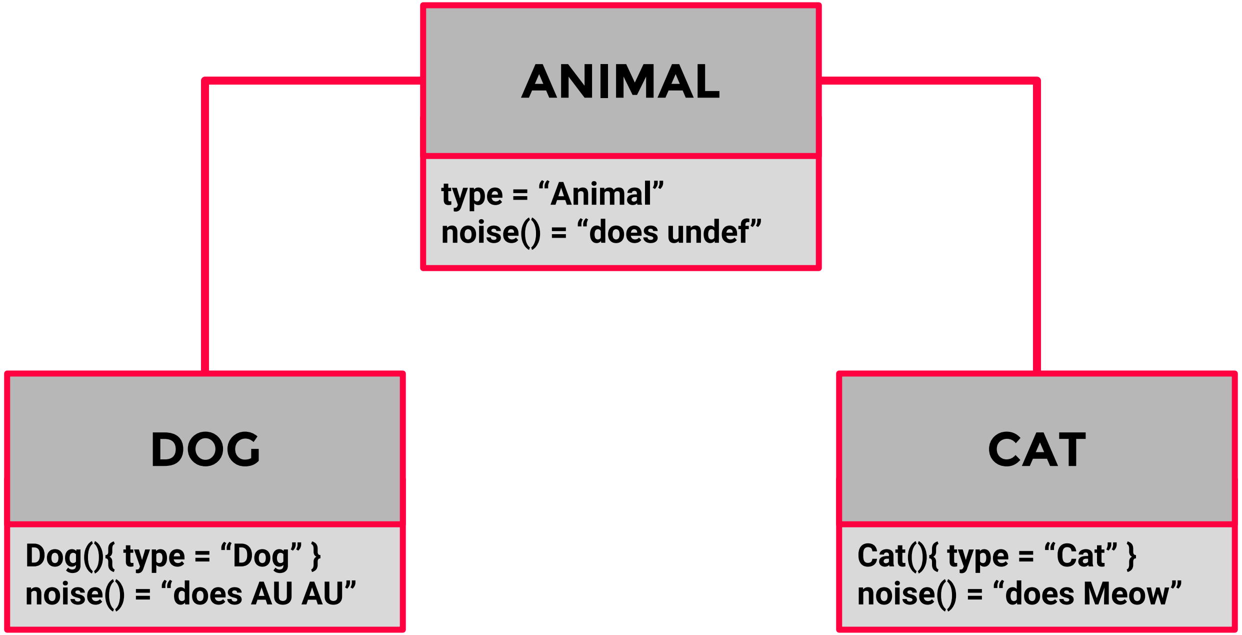
```
int main(){

        Animal* animal = new Dog;
        Cat* cat = static_cast<Cat*>(animal);

        cout << “I’m a cat, so:” << endl;
        cout << cat->type;
        cat->noise();


        return 0;
}
```

Code sample

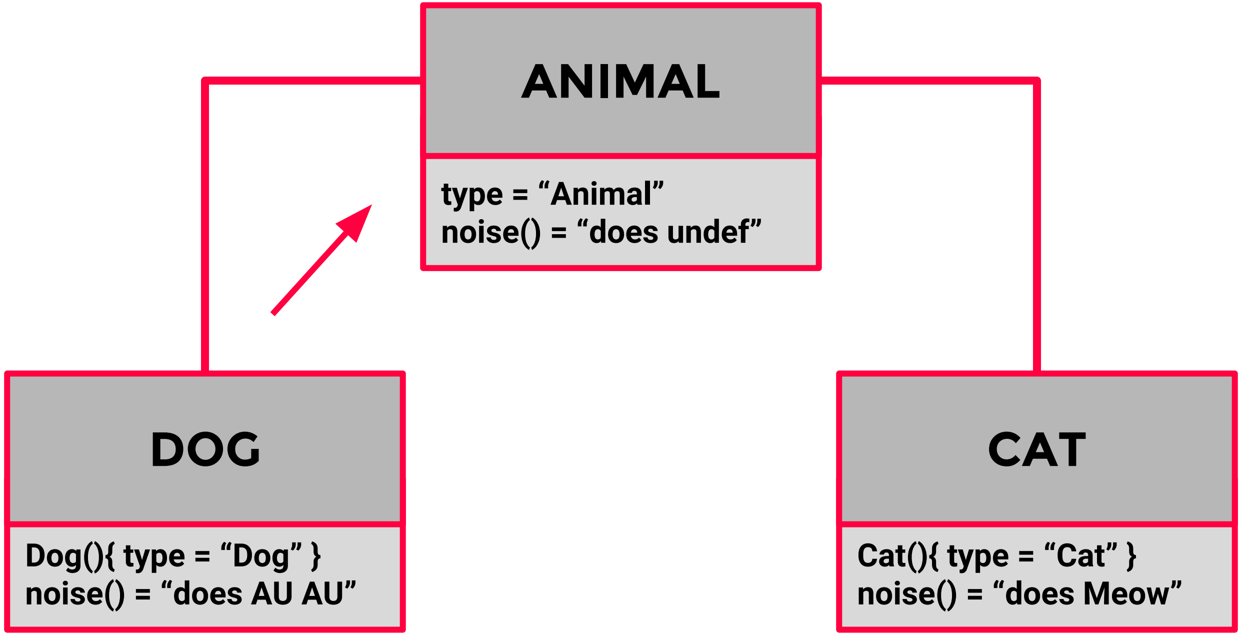
Output

```
int main(){

        Animal* animal = new Dog;
        Cat* cat = static_cast<Cat*>(animal);

        cout << “I’m a cat, so:” << endl;
        cout << cat->type;
        cat->noise();


        return 0;
}
```

Remembering the scenario



Remembering the scenario



ANIMAL

type = “Animal”
noise() = “does undef”

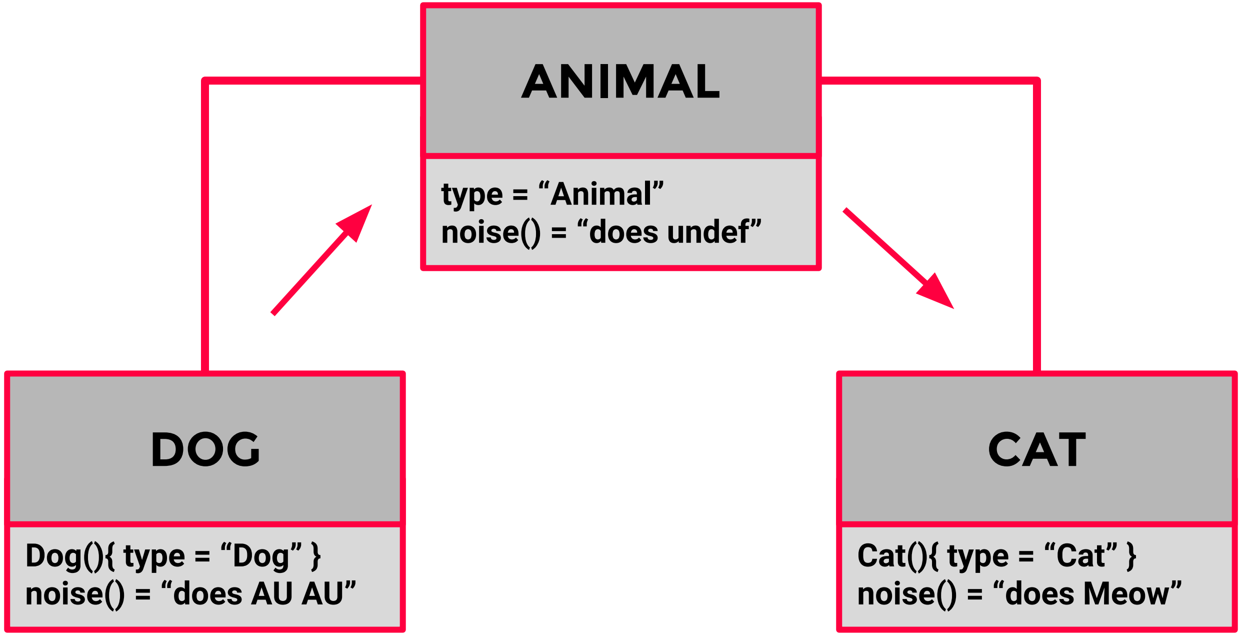
DOG

Dog(){ type = “Dog” }
noise() = “does AU AU”

CAT

Cat(){ type = “Cat” }
noise() = “does Meow”

Remembering the scenario



ANIMAL

type = “Animal”
noise() = “does undef”

DOG

Dog(){ type = “Dog” }
noise() = “does AU AU”

CAT

Cat(){ type = “Cat” }
noise() = “does Meow”

Code sample

```
int main(){

        Animal* animal = new Dog;
        Cat* cat = static_cast<Cat*>(animal);

        cout << “I’m a cat, so:” << endl;
        cout << cat->type;
        cat->noise();


        return 0;
}
```

Output

Code sample

Output

```
int main(){

        Animal* animal = new Dog;
        Cat* cat = static_cast<Cat*>(animal);

        cout << “I’m a cat, so:” << endl;
        cout << cat->type;
        cat->noise();


        return 0;
}
```

Code sample

```
int main(){

        Animal* animal = new Dog;
        Cat* cat = static_cast<Cat*>(animal);

        cout << “I’m a cat, so:” << endl;
        cout << cat->type;
        cat->noise();


        return 0;
}
```

Output

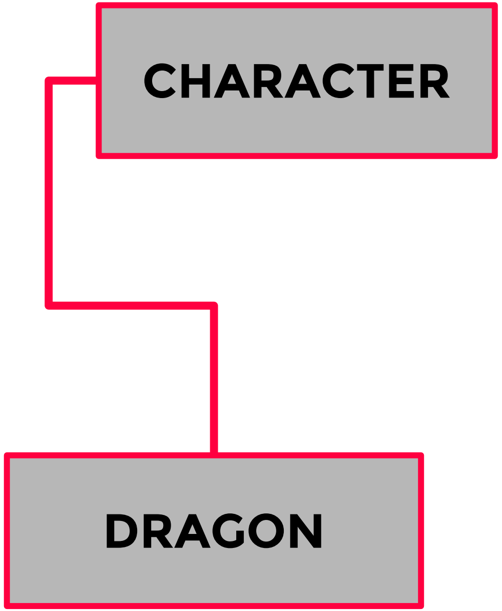
```
I’m a cat, so:
Dog does Meow
```

DYNAMIC_CAST

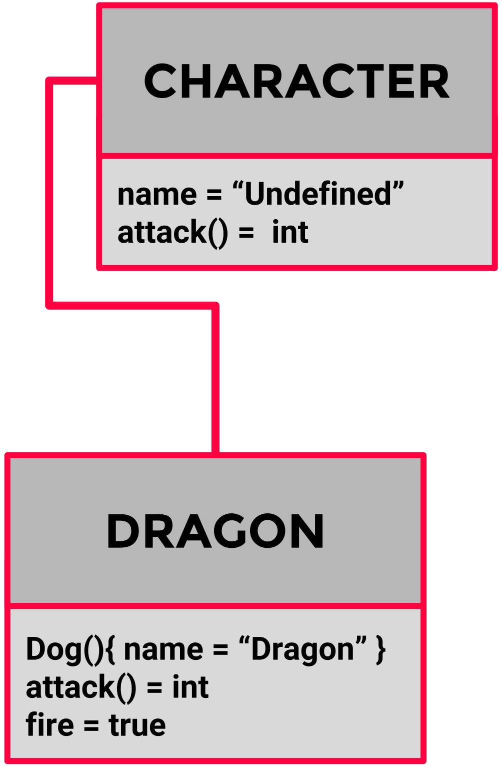
DYNAMIC CAST

Only with objects

Creating the scenario



Creating the scenario



CHARACTER

name = “Undefined”
attack() = int

DRAGON

Dog(){ name = “Dragon” }
attack() = int
fire = true

Code sample

Output

```
int main(){

    Character* charc = new Dragon;
    Dragon* dragon = dynamic_cast<Dragon*>(charc);






    return 0;
}
```

Code sample

Output

```
int main(){

        Character* charc = new Dragon;
        Dragon* dragon = dynamic_cast<Dragon*>(charc);

        cout << typeid(charc).name() << endl;
        cout << typeid(dragon).name() << endl;



        return 0;
}
```

Code sample

```
int main(){

	Character* charc = new Dragon;
	Dragon* dragon = dynamic_cast<Dragon*>(charc);

	cout << typeid(charc).name() << endl;
	cout << typeid(dragon).name() << endl;



	return 0;
}
```

Output

```
P9Character
P6Dragon
```

DYNAMIC CAST

Run-time type information

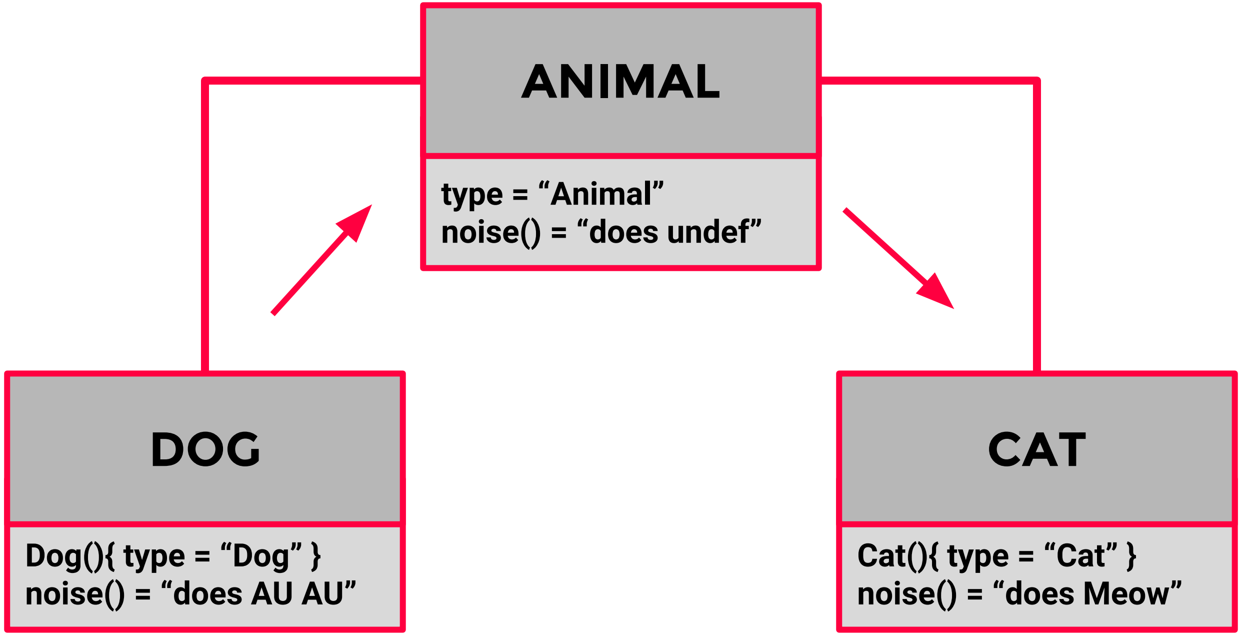
Code sample

```
int main(){

        Animal* animal = new Dog;
        Cat* cat = static_cast<Cat*>(animal);

        cout << “I’m a cat, so:” << endl;
        cout << cat->type;
        cat->noise();


        return 0;
}
```

Output

```
I’m a cat, so:
Dog does Meow
```

Remembering the scenario



ANIMAL

type = “Animal”
noise() = “does undef”

DOG

Dog(){ type = “Dog” }
noise() = “does AU AU”

CAT

Cat(){ type = “Cat” }
noise() = “does Meow”

Code sample

```
int main(){

    Animal* animal = new Dog;
    Cat* cat = static_cast<Cat*>(animal);

    cout << “I’m a cat, so:” << endl;
    cout << cat->type;
    cat->noise();


    return 0;
}
```

Output

```
I’m a cat, so:
Dog does Meow
```

Code sample

```
int main(){

    Animal* animal = new Dog;


    cout << “I’m a cat, so:” << endl;
    cout << cat->type;
    cat->noise();


    return 0;
}
```

Output

Code sample

Output

```
int main(){

    Animal* animal = new Dog;
    Cat* cat = dynamic_cast<Cat*>(animal);

    cout << “I’m a cat, so:” << endl;
    cout << cat->type;
    cat->noise();


    return 0;
}
```

Code sample

```
int main(){

    Animal* animal = new Dog;
    Cat* cat = dynamic_cast<Cat*>(animal);

    cout << “I’m a cat, so:” << endl;
    cout << cat->type;
    cat->noise();


    return 0;
}
```

Output

```
error: Segmentation fault
(core dumped)
```

Code sample

```
int main(){

    Animal* animal = new Dog;
    Cat* cat = dynamic_cast<Cat*>(animal);







    return 0;
}
```

Code sample

```
int main(){

    Animal* animal = new Dog;
    Cat* cat = dynamic_cast<Cat*>(animal);

    if(!cat)
        cout << "BAD CAST";



    return 0;
}
```

Output

Code sample

```cpp
int main(){

	Animal* animal = new Dog;
	Cat* cat = dynamic_cast<Cat*>(animal);

	if(!cat)
		cout << "BAD CAST";



	return 0;
}
```

Output

```
BAD CAST
```

Code sample

```
int main(){

    Animal* animal = new Dog;
    Cat* cat = dynamic_cast<Cat*>(animal);






    return 0;
}
```

Code sample

```
int main(){

        Animal* animal = new Cat;
        Cat* cat = dynamic_cast<Cat*>(animal);







        return 0;
}
```

Code sample

```
int main(){

        Animal* animal = new Cat;
        Cat* cat = dynamic_cast<Cat*>(animal);

        cout << “I’m a cat, so:” << endl;
        cout << cat->type;
        cat->noise();


        return 0;
}
```

Output

Code sample

```
int main(){

	Animal* animal = new Cat;
	Cat* cat = dynamic_cast<Cat*>(animal);

	cout << “I’m a cat, so:” << endl;
	cout << cat->type;
	cat->noise();


	return 0;
}
```

Output

```
I’m a cat, so:
Cat does Meow
```

DYNAMIC CAST

Substitute static_cast ?

Code sample

```
int main(){

        int number = dynamic_cast<int>(2.5);

        return 0;
}
```

Output

Code sample

```
int main(){

    int number = dynamic_cast<int>(2.5);

    return 0;
}
```

Output

```
error: cannot dynamic_cast
‘2.5e+0f’ (of type ‘float’)
to type ‘int’
```

REINTERPRET CAST

REINTERPRET CAST

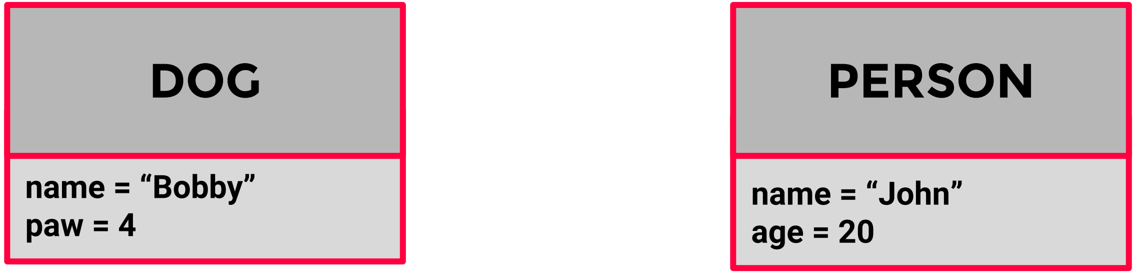
Converts any pointer type to any other pointer type

Creating the scenario

DOG

PERSON

Creating the scenario



DOG

name = “Bobby”
paw = 4

PERSON

name = “John”
age = 20

Code sample

```
int main(){

    Person* person = new Person;
    Dog* dog = reinterpret_cast<Dog*>(person);







    return 0;
}
```

Code sample

Output

```
int main(){

	Person* person = new Person;
	Dog* dog = reinterpret_cast<Dog*>(person);

	cout << “Name: ”;
	cout << dog->name << endl;
	cout << “Paw: ”;
	cout << dog->paw << endl;


	return 0;
}
```

Remembering the scenario

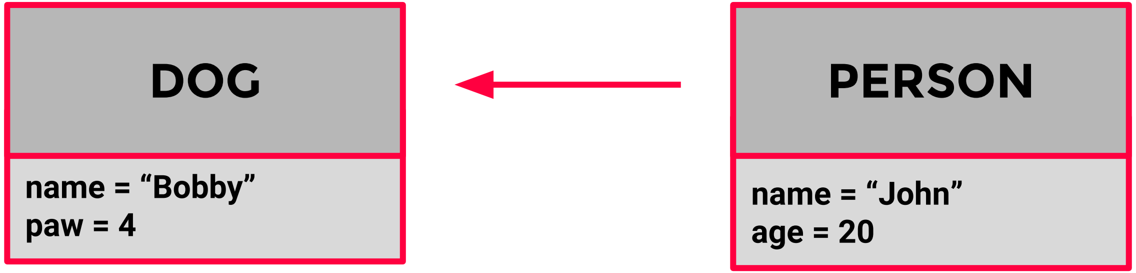
DOG

name = “Bobby”
paw = 4

PERSON

name = “John”
age = 20

Remembering the scenario



DOG

name = “Bobby”
paw = 4

PERSON

name = “John”
age = 20

Code sample

```
int main(){

    Person* person = new Person;
    Dog* dog = reinterpret_cast<Dog*>(person);

    cout << “Name: ”;
    cout << dog->name << endl;
    cout << “Paw: ”;
    cout << dog->paw << endl;


    return 0;

}
```

Output

Code sample

```
int main(){

        Person* person = new Person;
        Dog* dog = reinterpret_cast<Dog*>(person);

        cout << “Name: ”;
        cout << dog->name << endl;
        cout << “Paw: ”;
        cout << dog->paw << endl;


        return 0;

}
```

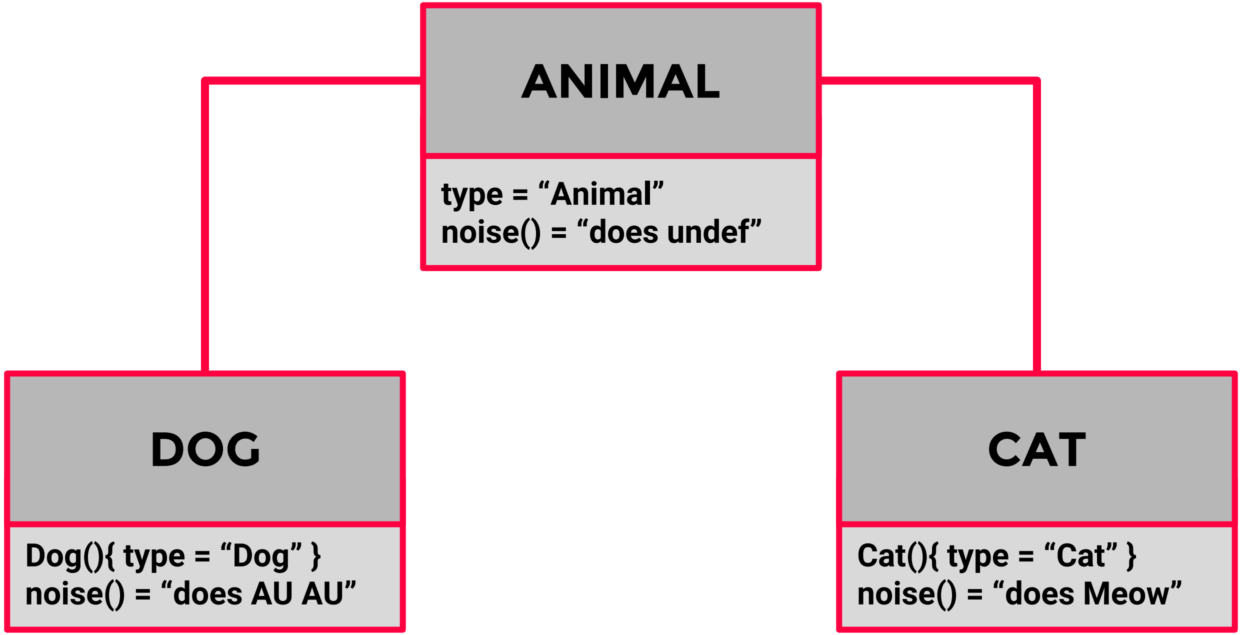
Output

```
Name: John
Paw: 20
```

REINTERPRET CAST

Does not do class hierarchy navigation

Remembering the scenario



ANIMAL

type = “Animal”
noise() = “does undef”

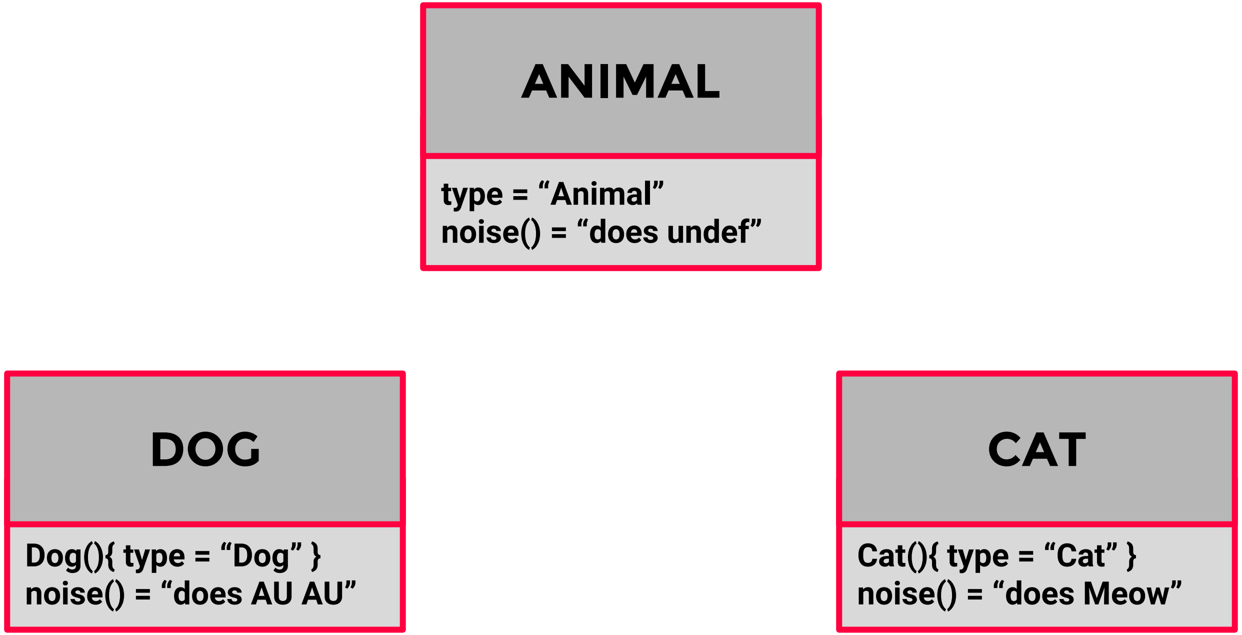
DOG

Dog(){ type = “Dog” }
noise() = “does AU AU”

CAT

Cat(){ type = “Cat” }
noise() = “does Meow”

Remembering the scenario



ANIMAL

type = “Animal”
noise() = “does undef”

DOG

Dog(){ type = “Dog” }
noise() = “does AU AU”

CAT

Cat(){ type = “Cat” }
noise() = “does Meow”

CONST CAST

Only changes cv-qualification

Code sample

```
int main(){
        char result[] = "7x1";

        const char* germany = result;



        return 0;
}
```

Code sample

```
int main(){
        char result[] = "7x1";

        const char* germany = result;
        cout << germany[0] << germany[1] << germany[2] <<
        endl;










        return 0;
}
```

Output

```
7x1
```

Code sample

```
int main(){
        char result[] = "7x1";

        const char* germany = result;
        cout << germany[0] << germany[1] << germany[2] <<
        endl;

        char* brazil = (char*) germany;









        return 0;
}
```

Output

```
7x1
```

Code sample

```
int main(){
        char result[] = "7x1";

        const char* germany = result;
        cout << germany[0] << germany[1] << germany[2] <<
        endl;

        char* brazil = (char*) germany;
        brazil[2] = '8';




        return 0;
}
```

Output

```
7x1
```

Code sample

```
int main(){
        char result[] = "7x1";

        const char* germany = result;
        cout << germany[0] << germany[1] << germany[2] <<
        endl;

        char* brazil = (char*) germany;
        brazil[2] = '8';
        cout << brazil[0] << brazil[1] << brazil[2] << endl;




        return 0;
}
```

Output

```
7x1
7x8
```

Code sample

```
int main(){
        char result[] = "7x1";

        const char* germany = result;
        cout << germany[0] << germany[1] << germany[2] <<
        endl;

        char* brazil = (char*) germany;
        brazil[2] = '8';
        cout << brazil[0] << brazil[1] << brazil[2] << endl;

        char* england = reinterpret_cast<char*>(germany);
        char* italy = dynamic_cast<char*>(germany);
        char* spain = static_cast<char*>(germany);

        return 0;
}
```

Output

```
error: reinterpret_cast from
type 'const char*' to type
'char*' casts away qualifiers
```

Code sample

Output

```
int main(){
        char result[] = "7x1";

        const char* germany = result;
        cout << germany[0] << germany[1] << germany[2] <<
        endl;

        char* brazil = (char*) germany;
        brazil[2] = '8';
        cout << brazil[0] << brazil[1] << brazil[2] << endl;




        return 0;
}
```

Code sample

```
int main(){
        char result[] = "7x1";

        const char* germany = result;
        cout << germany[0] << germany[1] << germany[2] <<
        endl;

        char* brazil = const_cast<char*>(germany);
        brazil[2] = '8';
        cout << brazil[0] << brazil[1] << brazil[2] << endl;




        return 0;
}
```

Output

```
7x1
7x8
```

Review

dynamic_cast<T>(e)
for run-time checked casts

static_cast<T>(e)
for reasonably well-behaved casts

reinterpret_cast<T>(e)
for casts yielding values that must be cast back to be used safely

const_cast<T>(e)
for casting away const

3.
Promotions and conversions in C++

Is quite simple

Promotions

A numeric promotion is the conversion of a value to a type with a wider range that happens whenever a value of a narrower type is used.

Conversions

A value can be numeric converted to another numeric type if required, but certain legal conversions can give different results using different compilers.

Is quite simple

Promotions/upcast

A numeric promotion is the conversion of a value to a type with a wider range that happens whenever a value of a narrower type is used.

Conversions/downcast

A value can be numeric converted to another numeric type if required, but certain legal conversions can give different results using different compilers.

A tip from Bjarne Stroustrup

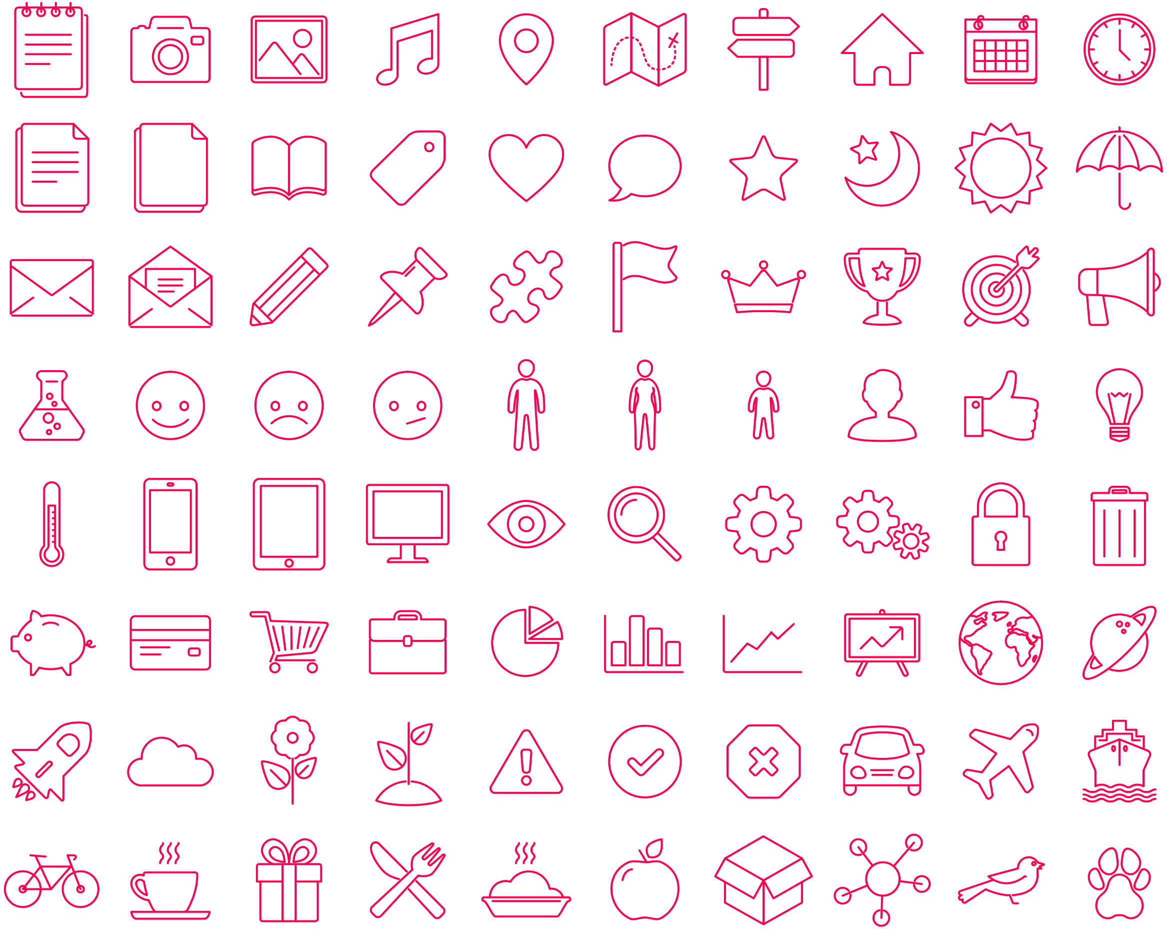
As ever, the moral of the story is: Avoid casts – of any sort – whenever possible.

Thanks!!

Any questions?

References

- New Casts Revisited by Bjarne Stroustrup
- Discussion at Quora Website



SlidesCarnival icons are editable shapes.

This means that you can:

- Resize them without losing quality.
- Change line color, width and style.

Isn’t that nice? :)

Examples: