#include <iostream>

#include <typeinfo>

using namespace std;

//class Animal

//{

//

//};

//

//class Cat : public Animal

//{

//

//};

//

//class Animal2

//{

// virtual void foo()

// {}

//

//};

//

//class Cat2 : public Animal2

//{

//

//};

//int main()

//{

// Cat cat;

// Animal& animal = cat;

//

// cout << "Reference to non-polymorphic " << typeid(animal).name() << endl;

//

// Cat2 cat2;

// Animal2& animal2 = cat2;

//

// cout << "Reference to polymorphic " << typeid(animal2).name() << endl;

//

// return 0;

//}

//int main()

//{

// int i = 5;

// float j = 1.5f;

// char c = 'a';

//

// const type\_info& t1 = typeid(i \* j);

// const type\_info& t2 = typeid(i \* c);

// const type\_info& t3 = typeid(c);

//

// cout << "t1 is of type " << t1.name() << endl;

// cout << "t2 is of type " << t2.name() << endl;

// cout << "t3 is of type " << t3.name() << endl;

//

//

//

//

// return 0;

//}

//

//void main()

//{/\*

// float f = 3.5f;

//

// int b = static\_cast<int>(f);

//

// cout << b << endl;\*/

//

// int a = 10;

// char c = 'a';

//

// int\* q = (int\*)&c;

//

// cout << \*q << endl;

//

// int\* p = static\_cast<int\*>(&c); // error

//

//

//

//

//}

#include <string>

//

//class INT

//{

// int x = 0;

//public:

// INT(int x = 0) :x(x)

// {

// cout << "Conversion ctor called" << endl;

// }

//

// int GetX() const

// {

// return x;

// }

//

// operator string()

// {

// cout << "Conversion operator called" << endl;

// return to\_string(x);

// }

//};

//

//void main()

//{

// INT obj(123);

// string str = obj;

// cout << str << endl;

//

// string str2 = static\_cast<string>(obj);

// cout << str2 << endl;

//

// INT obj2 = static\_cast<INT>(100);

// cout << obj2.GetX() << endl;

//}

//

//class Base

//{

//protected:

// int m\_value;

//public:

// Base(int m\_value)

// {

// this->m\_value = m\_value;

// }

// virtual ~Base()

// {

//

// }

//};

//

//class Derived : public Base

//{

//protected:

// string name;

//public:

// Derived(int value, string name)

// : Base(value), name(name) {}

//

// string GetName() const

// {

// return name;

// }

//};

//

//Base\* getObject(bool returnDerived)

//{

// if (returnDerived)

// {

// return new Derived(10, "apple");

// }

// else

// {

// return new Base(2);

// }

//}

//

//class Cat {};

//

//void main()

//{

// Base\* b = getObject(false);

//

// Derived\* d = dynamic\_cast<Derived\*>(b);

//

// if (d != nullptr)

// {

// cout << " Value of the class : " << d->GetName() << endl;

// }

// else

// {

// cout << "Can not cast : error NULL" << endl;

// }

//}

//

//class Animal

//{

//public:

// virtual void Speak()

// {

// cout << "???????????" << endl;

// }

//};

//

//class Cat: public Animal

//{

//public:

// void Speak() override

// {

// cout << "meauuuu" << endl;

// }

//

// void Run()

// {

// cout << "I run like cat" << endl;

// }

//};

//

//class Dog : public Animal

//{

//public:

// void Speak() override

// {

// cout << "meauuuu" << endl;

// }

//

// void Bark()

// {

// cout << "I bark like big dog " << endl;

// }

//};

//

//void main()

//{

// auto animals = new Animal \* [3]

// {

// new Cat,

// new Dog,

// new Cat

// };

//

// for (size\_t i = 0; i < 3; i++)

// {

//

// }

//

//}

//class Student

//{

// int roll;

// mutable int roll2; // const daxilinde deyismek ucun mutable istifade olunu

//public:

// Student(int r) : roll(r)

// {

//

// }

//

// void Fun() const

// {

// //this->roll = 20 // error

// this->roll2 = 200; // okay because roll2 is mutable

//

// (const\_cast<Student\*>(this))->roll = 20;

// }

//};

class Cat

{

public:

char data;

};

void main()

{

int\* p = new int(65);

Cat\* ch = reinterpret\_cast<Cat\*>(p);

int\* pp = reinterpret\_cast<int\*>(ch);

cout << \*p << endl;

cout << (\*ch).data << endl;

cout << \*pp << endl;

}