~ ÖDEV 5 ~

a) Burada swap fonk. parametrelerin kopyalarıyla çalışır. Bu yüzden orijinal değerler değişmez.

```
swap(value, list[0]);
swap(list[0], list[1]);
swap(value, list[value]);
```

value	list[]
2	{1,3,5,7,9}

b) Burada artık değişkenlerin asıl bellek adresleri temsil ediliyor. Dolayısıyla değişiklikler çağıran fonk. yansır.

swap(value, list[0]) → swap(2,1);

value = 1 list = {2,3,5,7,9};

swap(list[0], [1]); → swap(1,3);

value = 2 list = {3,1,5,7,9};

swap(value, list[value]); → swap(2,5);

value = 5 list = {1,3,2,7,9};

c) Bu örnekte parametreler kaster kopyaların (by-value gibi) ona fonksiyon bitince kopyaların son halleri geri yazılır.

swap(value, list[0]); → swap(2,1)

value = 1 list = {2,3,5,7,9}

swap(list[0], list[1]) → swap(1, 3)

value = 2 list = {3, 1, 5, 7, 9}

swap(value, list[value]) → swap(2, 5)

value = 5 list = {1, 5, 2, 7, 9}

	value	list
a. by value	2	{1, 3, 5, 7, 9}
b. by reference	5	{1, 3, 2, 7, 9}
c. by value-result	5	{1, 3, 2, 7, 9}

SORU 23 :

— Java ve C#'taki interface'ler sadece imza (signature) içerir. Yani; metot isimleri, geri dönüş türleri, parametreler. Ama metotların gövdesi yoktur (Bazı özel durumlar hariç). Bu nedenle, interfaceler sadece bir sözleşme (contract) gibidir. C++'ta ise bir sınıf birden fazla sınıftan hem davranış hem veri (kod) miras alır. Bu da karışıklık oluşturur.

— Elmas Problemi:
Çoklu kalıtımın en büyük problemi, aşğıda örneği verilen yapıda ortaya çıkar:

```
class A { public: void hello(); };
class B : public A {};
class C : public A {};
class D : public B, public C {};      // Problem
```

Burada D sınıfı hem B hem C üzerinden iki farklı A örneği alır. Bu durumda D içinde hello() fonksiyonunu çağırmak belirsizliğe (ambiguity) yol açar. (Hangi A'nın hello() fonksiyonu kullanılacak?)
Çözüm: C++'ta virtual inheritance gibi karmaşık çözümler gerektirir.

— Java ve C#'ta interface uygulamak sadece imza çakışmalarında sorun çıkarabilir. Örneğin;

```
interface A { void print(); }
interface B { void print(); }
class MyClass implements A, B {
    public void print() {
        System.out.println("Tek print yeterli.");
    }
}
```

Her iki interface de print() metodu tanımlasa bile gövdesi olmadığı için, MyClass sadece bir tane print() yazar ve bu yeterlidir. Yani çakışma yaşanmaz, çünkü aynı imza tek bir implemantasyonla çözülür.

SONUÇ:
Java ve C#'ta bir sınıfın birden fazla interfacei uygulayabilmesi, interfacelerin sadece metod imzası tanımlaması ve gövde içermemesi sayesinde belirsizlik oluştrmaz.
Bu yüzden, C++'taki çoklu kalıtım problemleri (özellikle Elmas Problemi) bu dillerde ortaya çıkmaz