Напишите функцию tee, которая направляет данные из канала `in` одновременно в два возвращаемых канала (т.е. в них попадают одинаковые данные), пока канал `in` открыт и контекст не отменен. Подсказка: для упрощения кода используйте `orDone` из прошлой задачи.

```
package main

import (
 "context"
 "reflect"
)

func main() {
 ctx, cancel := context.WithCancel(context.Background())
 defer cancel()
 i := 0
 inc := func() interface{} {
  i++
  return i
 }

 out1, out2 := tee(ctx, take(ctx, repeatFn(ctx, inc), 3))
 var res1, res2 []interface{}
 for val1 := range out1 {
  res1 = append(res1, val1)
  res2 = append(res2, <-out2)
 }
 exp := []interface{}{1, 2, 3}
 if !reflect.DeepEqual(res1, exp) || !reflect.DeepEqual(res2, exp) {
  panic("wrong code")
 }
}

func tee(ctx context.Context, in <-chan interface{}) (_, _ <-chan interface{}) {
 // напишите ваш код здесь
}

func orDone(ctx context.Context, in <-chan interface{}) <-chan interface{} {
 out := make(chan interface{})
 go func() {
  defer close(out)
  for {
   select {
   case <-ctx.Done():
    return
   case v, ok := <-in:
    if !ok {
     return
    }
    select {
    case out <- v:
    case <-ctx.Done():
    }
   }
  }
 }()
```

```
 return out
}

func repeatFn(ctx context.Context, fn func() interface{}) <-chan interface{} {
 out := make(chan interface{})
 go func() {
  defer close(out)
  for {
   select {
   case <-ctx.Done():
    return
   case out <- fn():
   }
  }
 }()
 return out
}

func take(ctx context.Context, in <-chan interface{}, num int) <-chan interface{} {
 out := make(chan interface{})
 go func() {
  defer close(out)
  for i := 0; i < num; i++ {
   select {
   case <-ctx.Done():
    return
   case v, ok := <-in:
    if !ok {
     return
    }
    out <- v
   }
  }
 }()
 return out
}
```