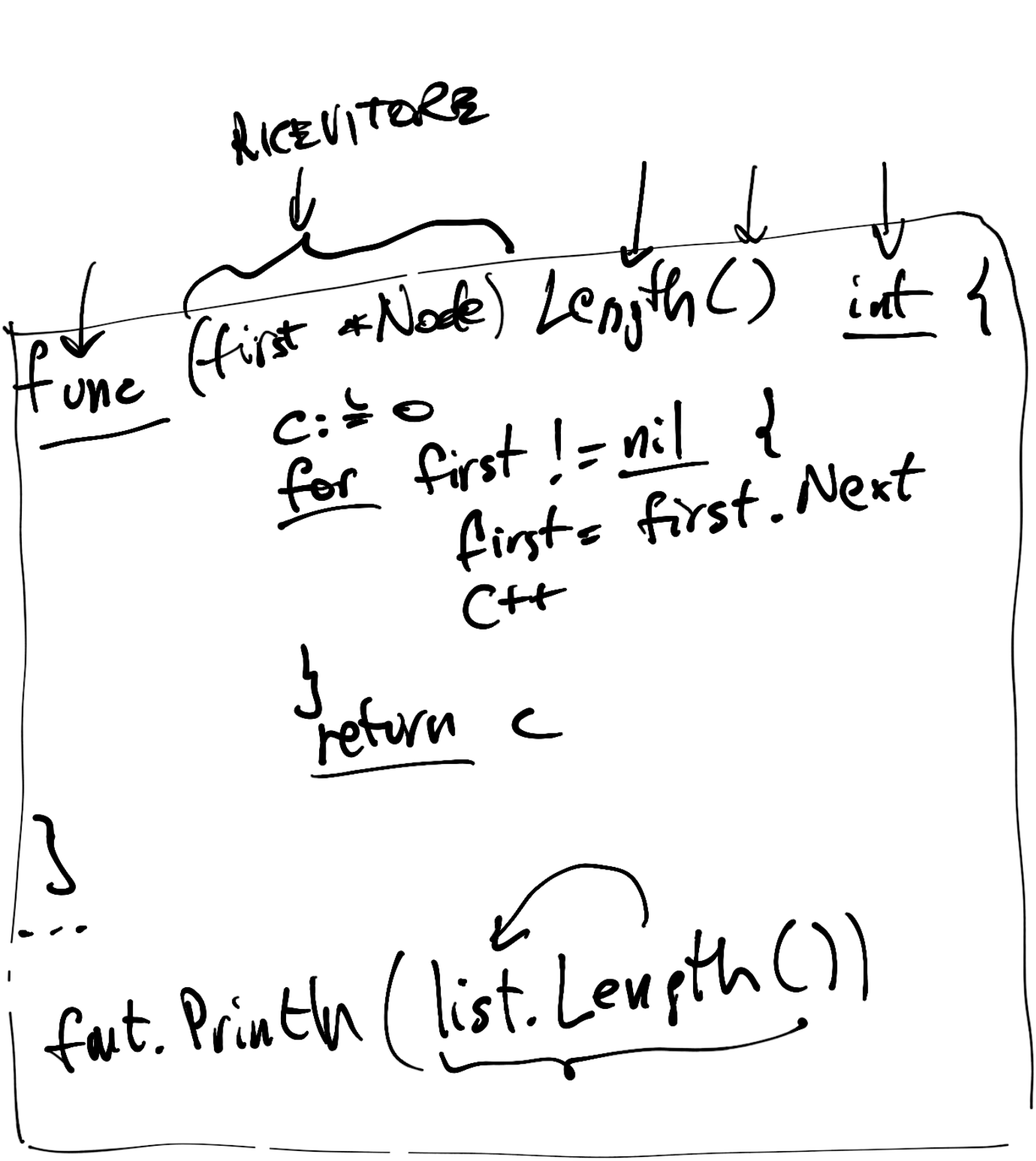
ADT = Abstract Data Type

TIPO DI DATO + FUNZIONI DI MANIPOL.

METODI

```
type Node struct {
    Data string
    Next *Node
}

func Length(first *Node) int {
    c := 0
    for first != nil {
        first = first.Next
        c++
    }
    return c
}
...
fmt.Println(Length(list))
```


RICEVITORE
func (first *Node) Length() int {
c := 0
for first != nil {
first = first.Next
c++
}
return c
}
...
fmt.Println(list.Length())

```
func (first *Node) AddFront(item string)
                *Node {
    n := new(Node)
    n.Next = first
    n.Data = item
    return n
}
```

---

list = AddFront(list, "pippo") PPUR

list = list.AddFront("pippo") ORA

FUNZIONI DI FABBRICAZIONE

scanner := bufio.NewScanner(os.Stdin)

for scanner.Scan() {

S....

}

fmt.Println

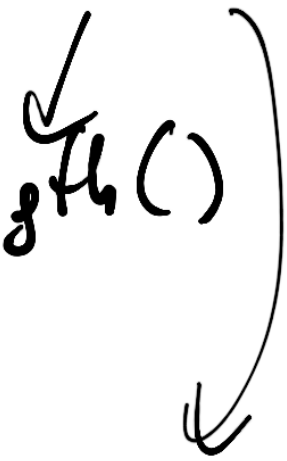
```
type Node {
    ...
}

func (first *Node) length() {
    ...
}

func (first *Node) AddFront(item ...) {
    ...
}

...
```

METODI DEL TIPO *Node



```
type Rettangolo [2]float64

func (r Rettangolo) Area() float64 {
    return r[0] * r[1]
}

func (r Rettangolo) Perimetro() float64 {
    return 2 * r[0] + 2 * r[1]
}

func (r Rettangolo) String() string {
    return fmt.Sprintf("base = %f, altezza = %f", r[0], r[1])
}
```

```
func NewRettangolo(b, h float64)
                        Rettangolo {

    var r Rettangolo
    r[0] = b
    r[1] = h
    return r

}
```

```
r := NewRettangolo(5, 7)
fmt.Println(r.Area())
fmt.Println(r)
```

```
type Cerchio struct {
    raggio float64
}

func (c Cerchio) Area() float64 {
    return c.raggio * c.raggio * math.Pi
}

func (c Cerchio) Perimetro() float64 {
    return 2.0 * math.Pi * c.raggio
}

func (c Cerchio) String() string {
    return fmt.Sprintf("raggio=%f",
        c.raggio)
}
```

```
func NewCerchio(raggio float64)
                    Cerchio {
    var c Cerchio
    c.raggio = raggio
    return c
}

func (c Cerchio) Circonferenza()
                    float64 {
    return c.Perimetro()
}
```

INTERFACCIA

Insieme di metodi che un tipo può implementare

```
type Figura interface {
    Area() float64
    Perimetro() float64
}
```

Si dice che Rettangolo e Cerchio implementano Figura

```
var f []Figura
f = append(f, New Rettangolo(3, 5))
f = append(f, New Cerchio(4))
f = append(f, New Rettangolo(7, 12))

avgArea := 0.0
for _, x := range f {
    avgArea += x.Area()
}
avgArea /= float64(len(f))
fmt.Println(avgArea)
```

var f Figura

→ f = New Cerchio (6)

f.Circonferenza() ↵ ?

---

type Stringer interface {

String() string

}

```
type Any interface {}

var x interface {}

x = 3
x = NewRettangolo(7,4)
x = "pippo"
```

```
package fmt

func Println(x interface{}...)
func Printf(s string, x interface{}...)
```