



UNIVERSITÀ
degli STUDI
di CATANIA

Fondamenti di Informatica

Liste di Puntatori

Maurizio Palesi

maurizio.palesi@dieei.unict.it

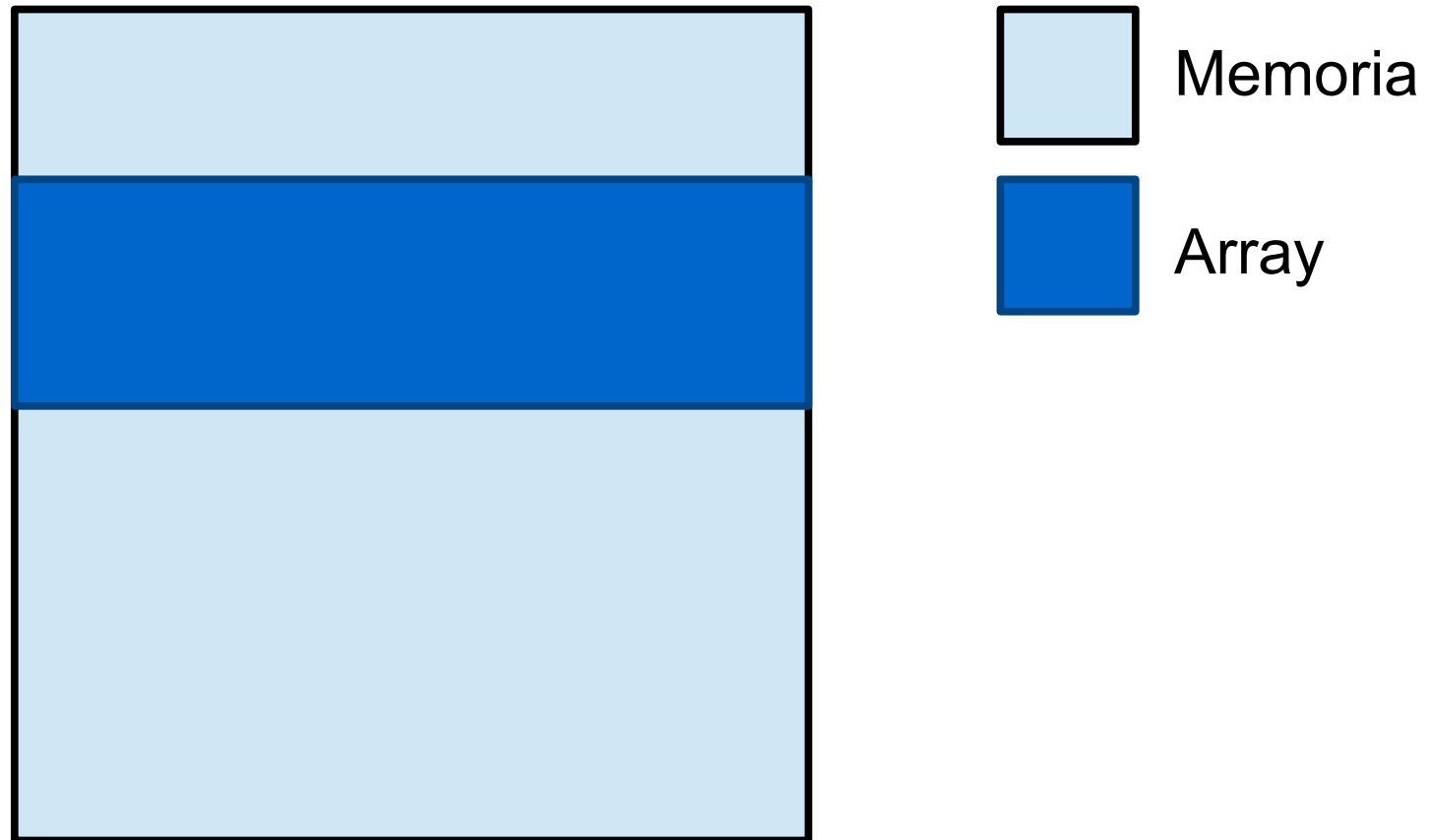
Osservazione sugli Array

- Gli elementi in un array sono mappati in **locazioni di memoria contigue**
- Pro
 - Efficienza nelle operazioni di trasferimento di dati da una parte della memoria ad un'altra
 - Accesso diretto agli elementi dell'array
- Contro
 - Difficoltà di allocazione se la memoria è frammentata
 - Operazioni di inserimento ed eliminazione mantenendo l'ordinamento delle informazioni poco efficiente
 - É richiesto lo scorrimento fisico dei dati

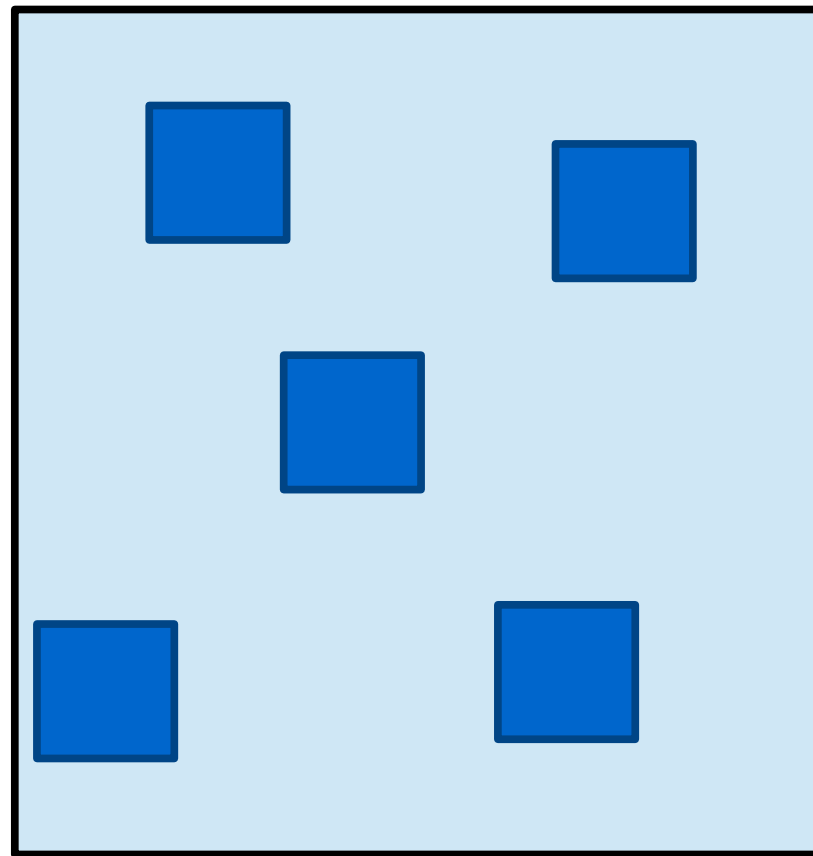
Liste di Puntatori

- I punti di debolezza degli array diventano punti di forza delle liste e viceversa
- In una lista le informazioni sono *sparse* nella memoria
 - Ogni elemento della lista deve contenere un *riferimento* all'elemento successivo (e/o precedente)
 - Tale riferimento rappresenta un *overhead*

Rappresentazione in Memoria – Array



Rappresentazione in Memoria – Lista

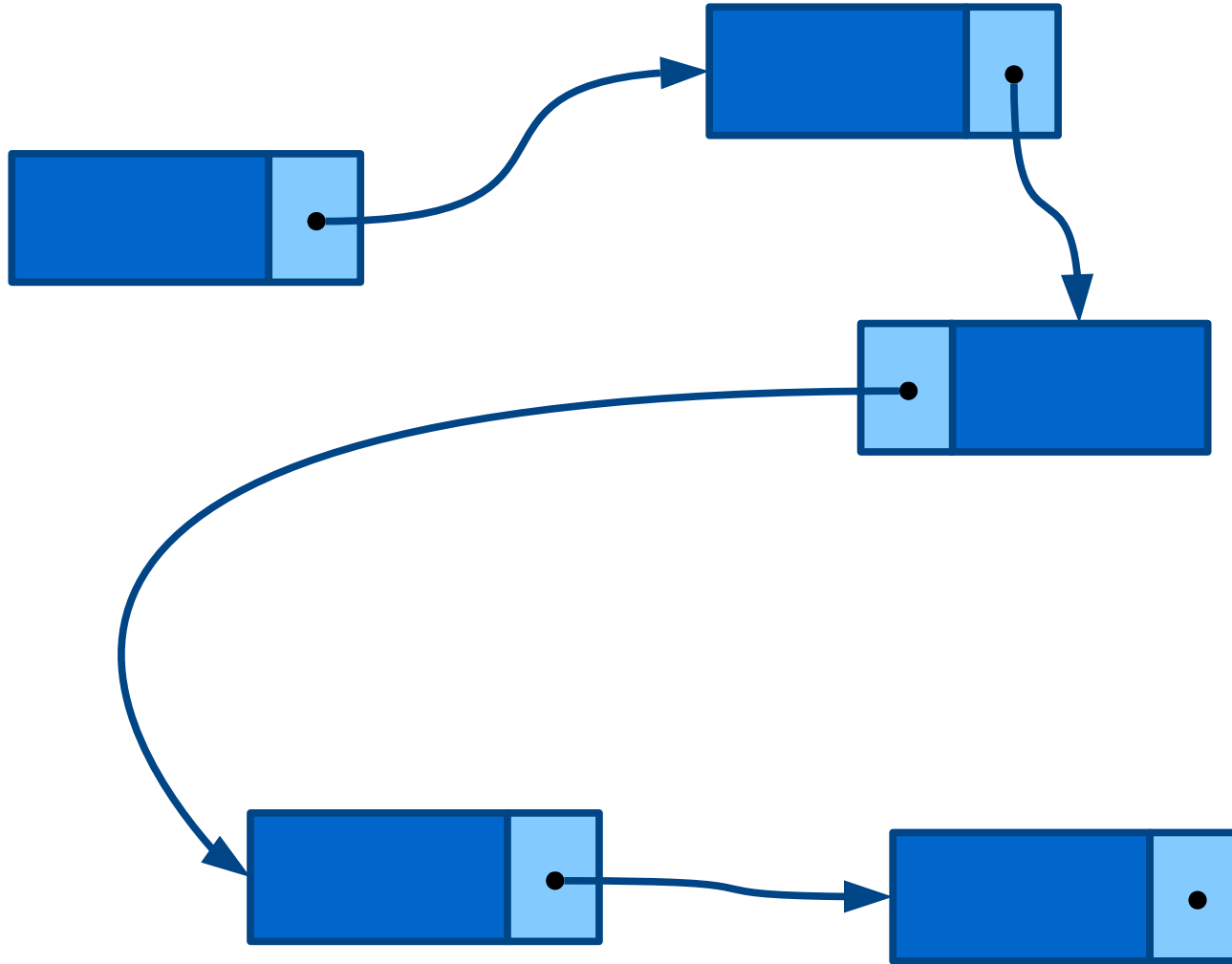


Memoria

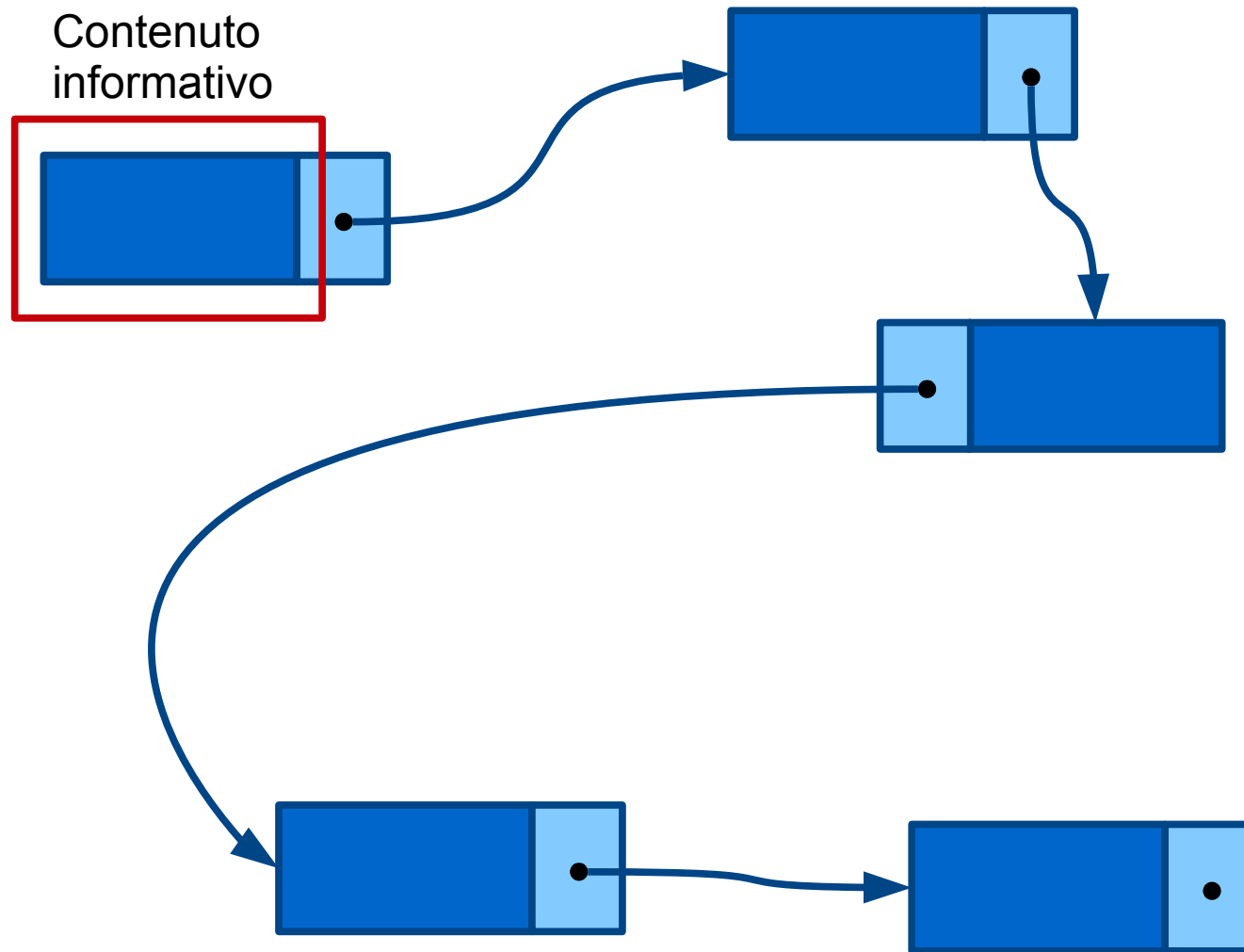


Nodo della lista

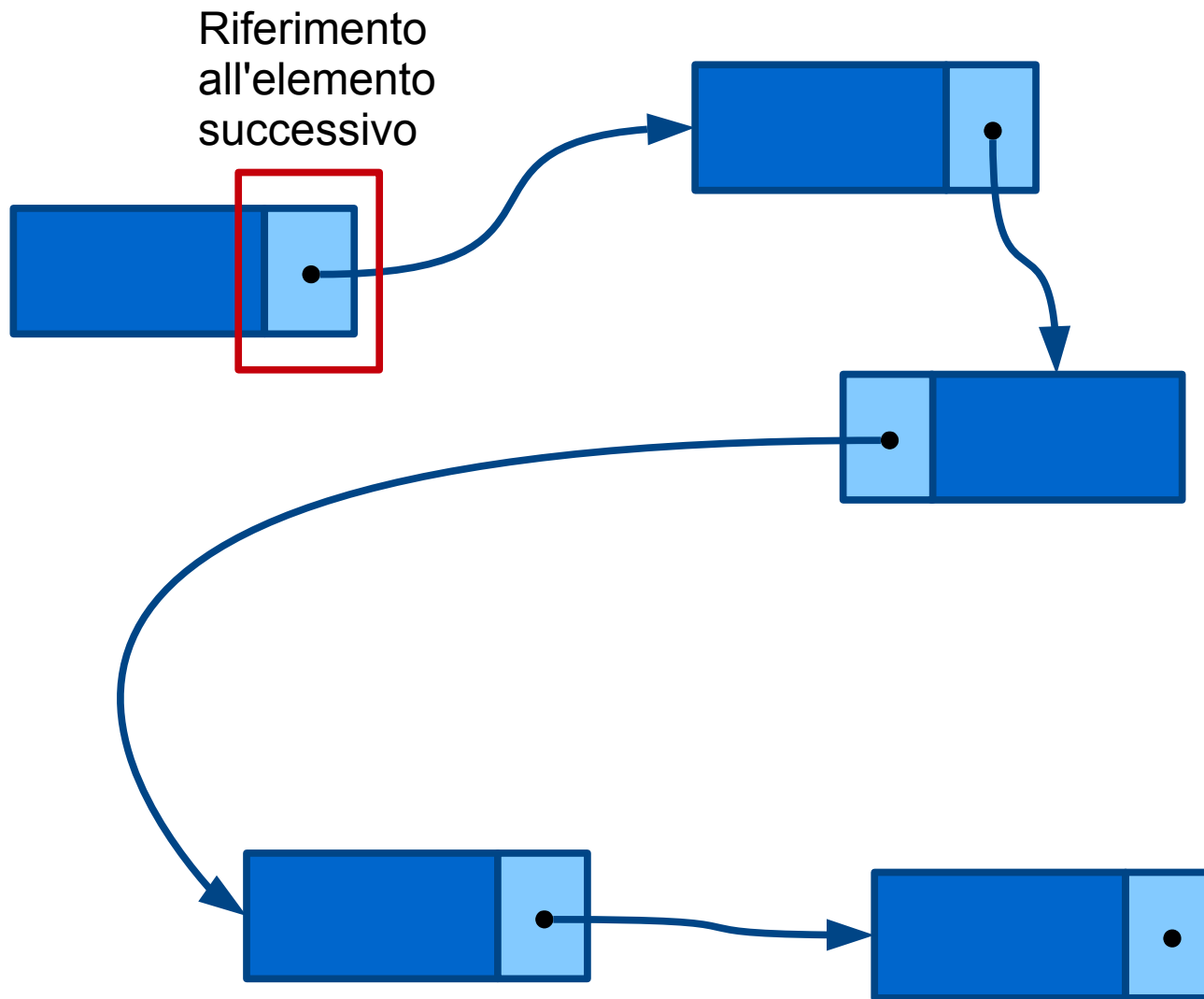
Rappresentazione in Memoria – Lista



Rappresentazione in Memoria – Lista



Rappresentazione in Memoria – Lista

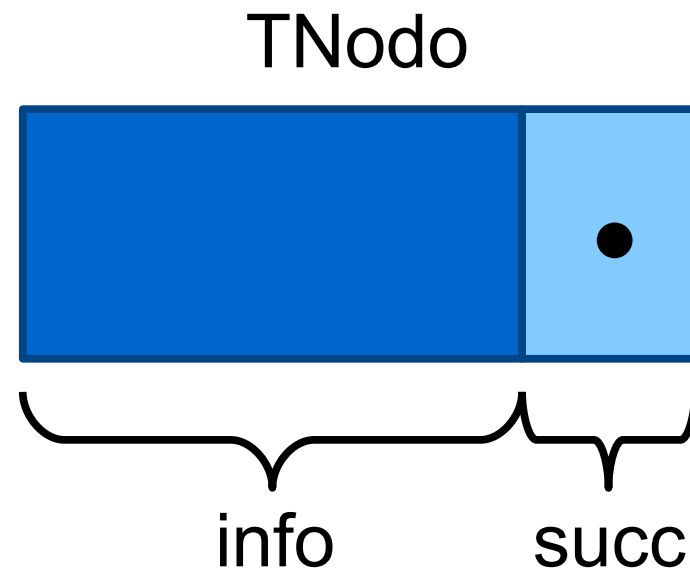


Definizione della Lista

```
typedef struct Nodo
{
    int      info;
    struct Nodo* succ;
} TNode;
```

Definizione della Lista

```
typedef struct Nodo
{
    int      info;
    struct Nodo* succ;
} TNode;
```



Scorrimento della Lista

```
typedef struct Nodo
{
    int          info;
    struct Nodo* succ;
} TNode;
```

```
typedef TNode* PNode;
```

```
PNode testa, p;
```

```
...
```

```
p = testa;
while (p != NULL) {
    ...
    p = p->succ;
}
```

Scorrimento della Lista

```
typedef struct Nodo
{
    int          info;
    struct Nodo* succ;
} TNode;
```

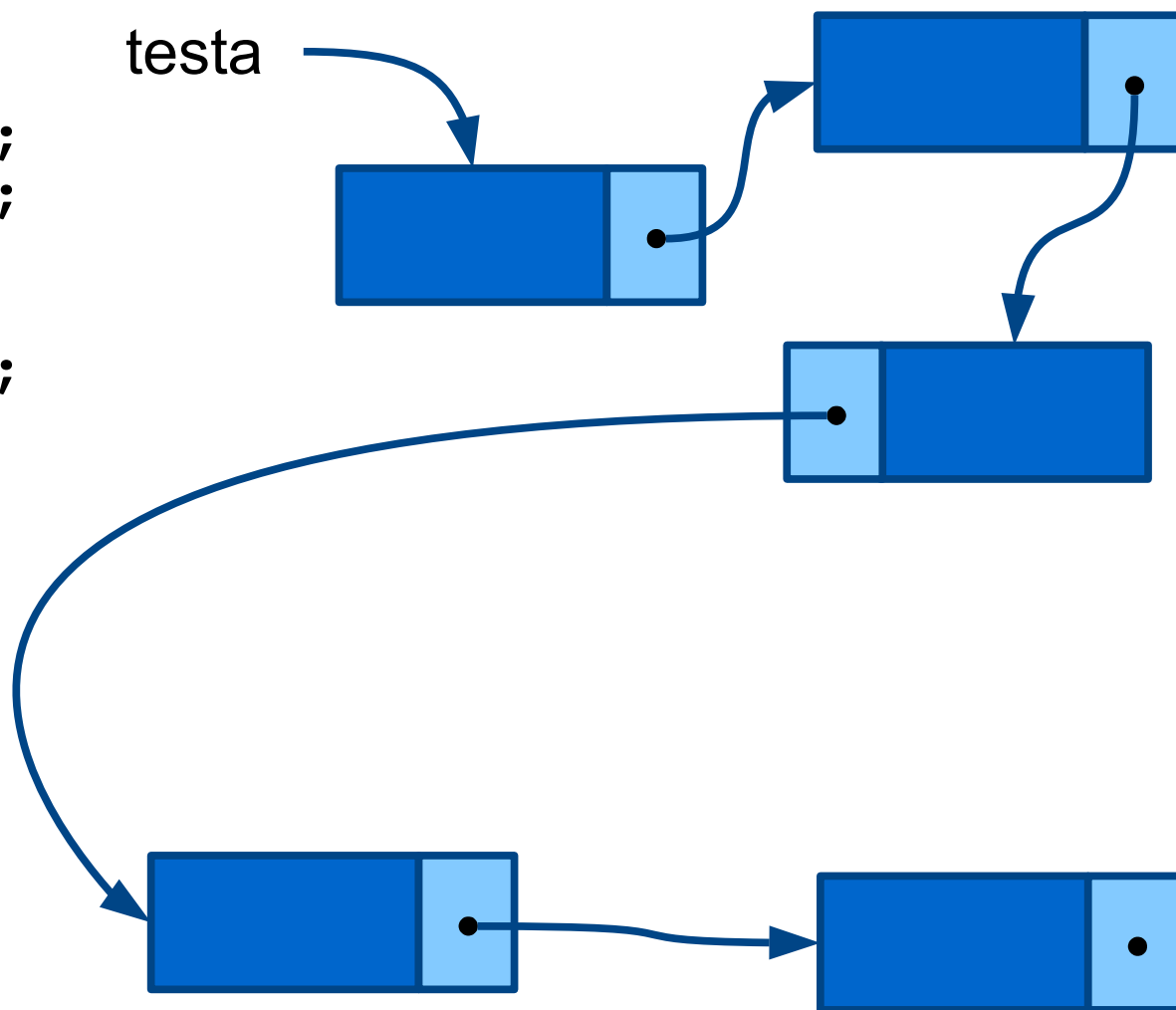
```
typedef TNode* PNode;
```

```
PNode testa, p;
```

```
...
```

➔

```
p = testa;
while (p != NULL) {
    ...
    p = p->succ;
}
```



Scorrimento della Lista

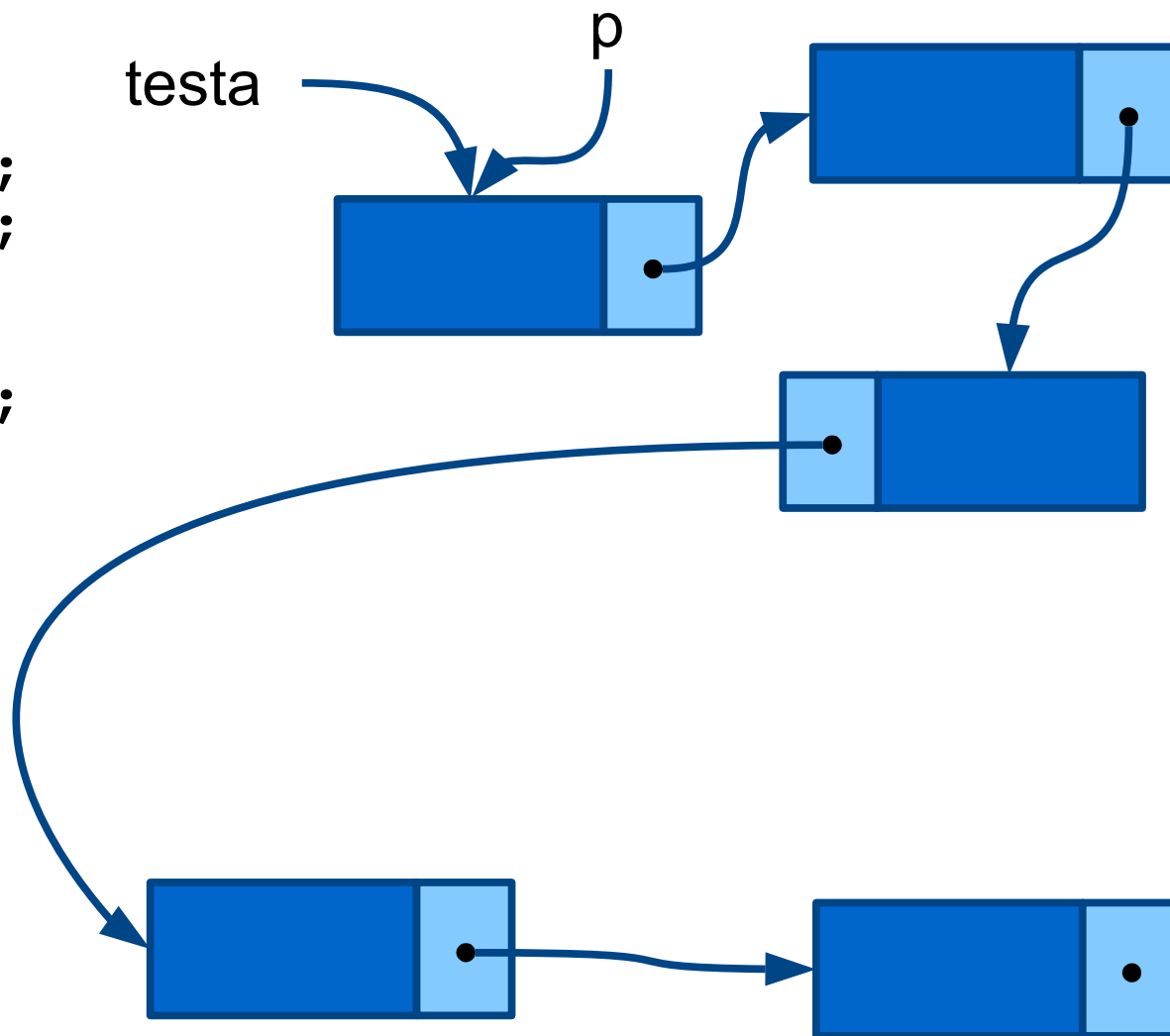
```
typedef struct Nodo
{
    int          info;
    struct Nodo* succ;
} TNode;
```

```
typedef TNode* PNode;
```

```
PNode testa, p;
```

```
...
```

```
➔ p = testa;
while (p != NULL) {
    ...
    p = p->succ;
}
```



Scorrimento della Lista

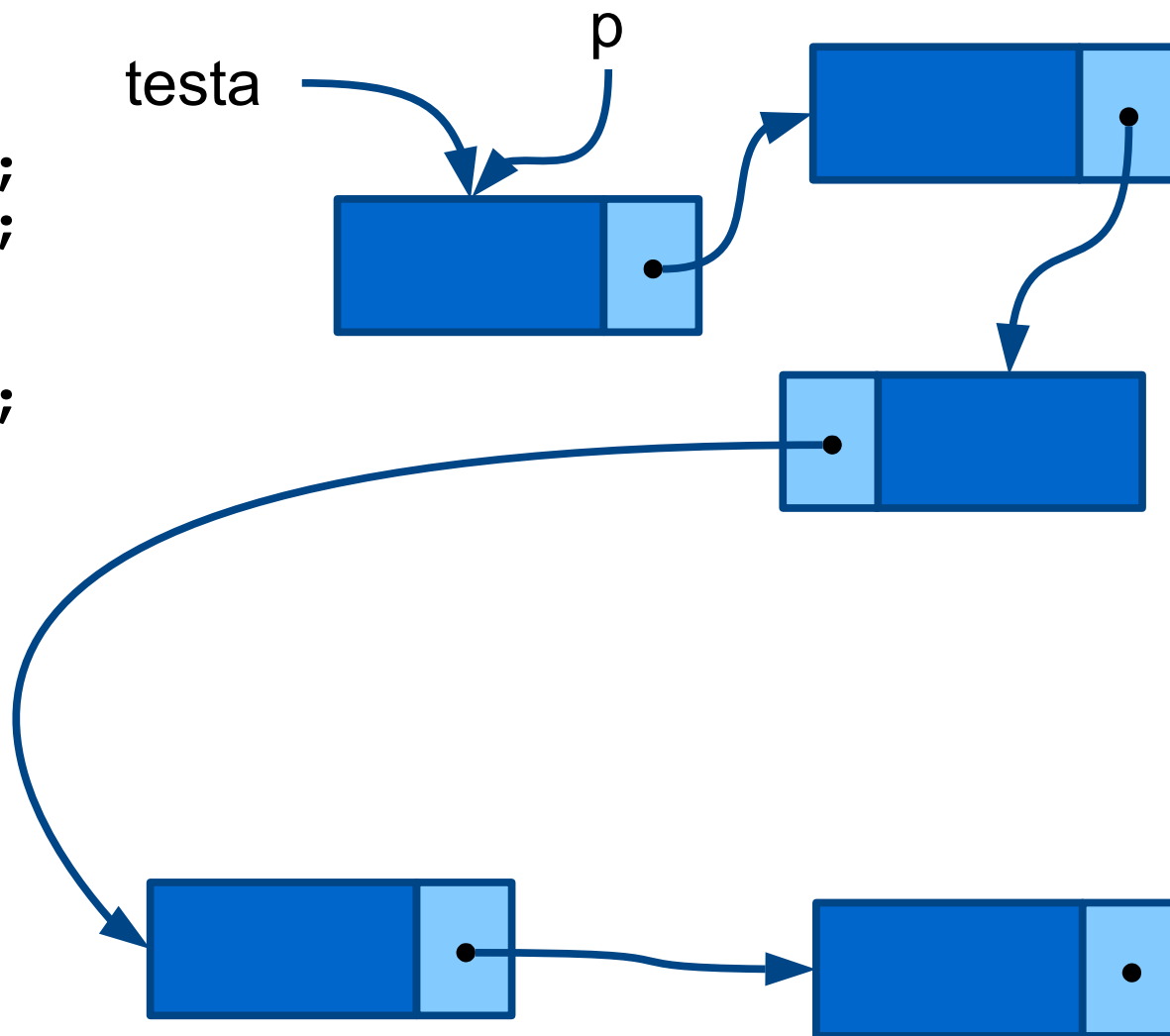
```
typedef struct Nodo
{
    int          info;
    struct Nodo* succ;
} TNode;
```

```
typedef TNode* PNode;
```

```
PNode testa, p;
```

```
...
```

```
p = testa;
while (p != NULL) {
    ...
    p = p->succ;
}
```



Scorrimento della Lista

```
typedef struct Nodo
{
    int          info;
    struct Nodo* succ;
} TNode;
```

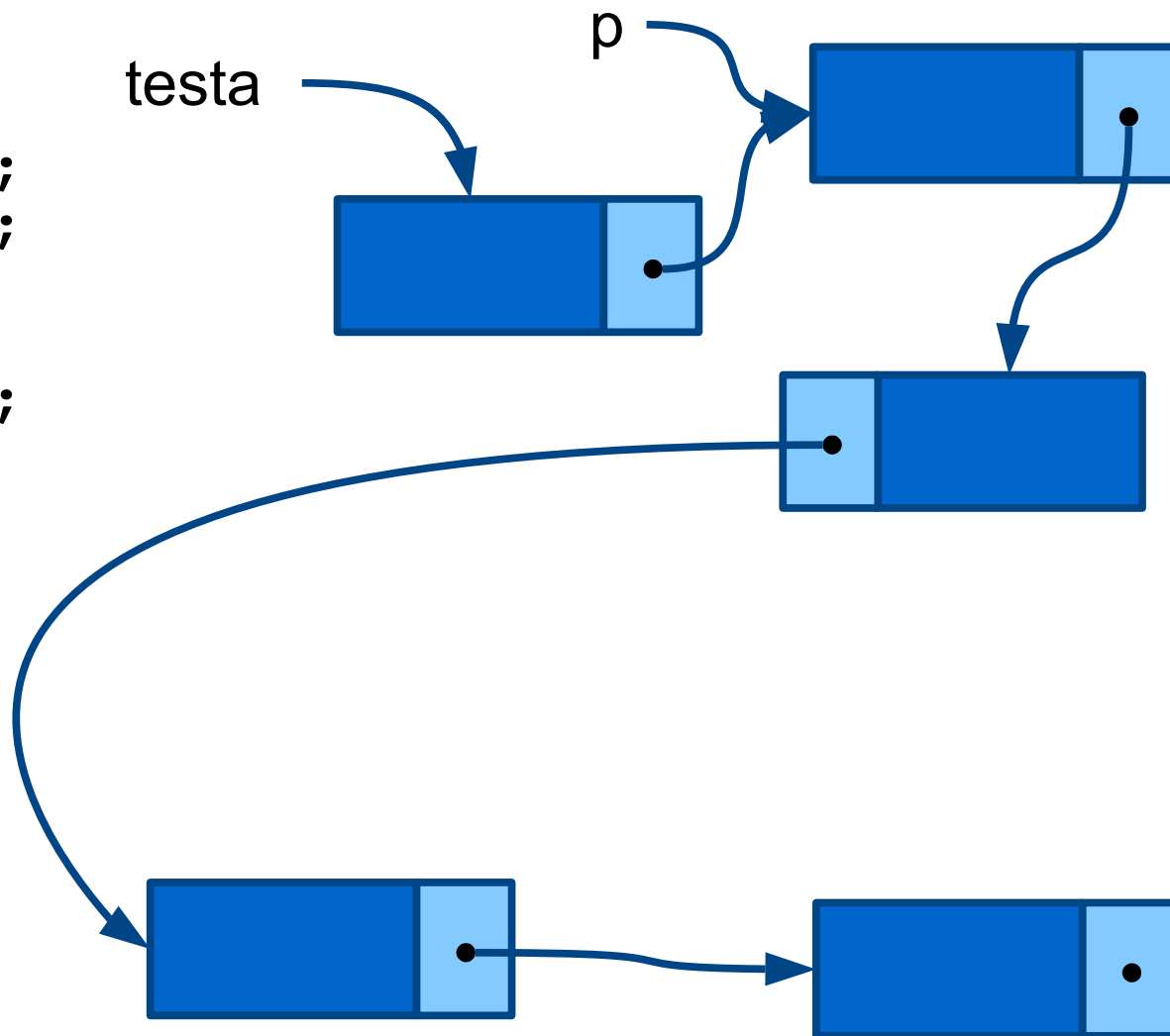
```
typedef TNode* PNode;
```

```
PNode testa, p;
```

```
...
```

```
p = testa;
```

```
→ while (p != NULL) {  
    ...  
    p = p->succ;  
}
```



Scorrimento della Lista

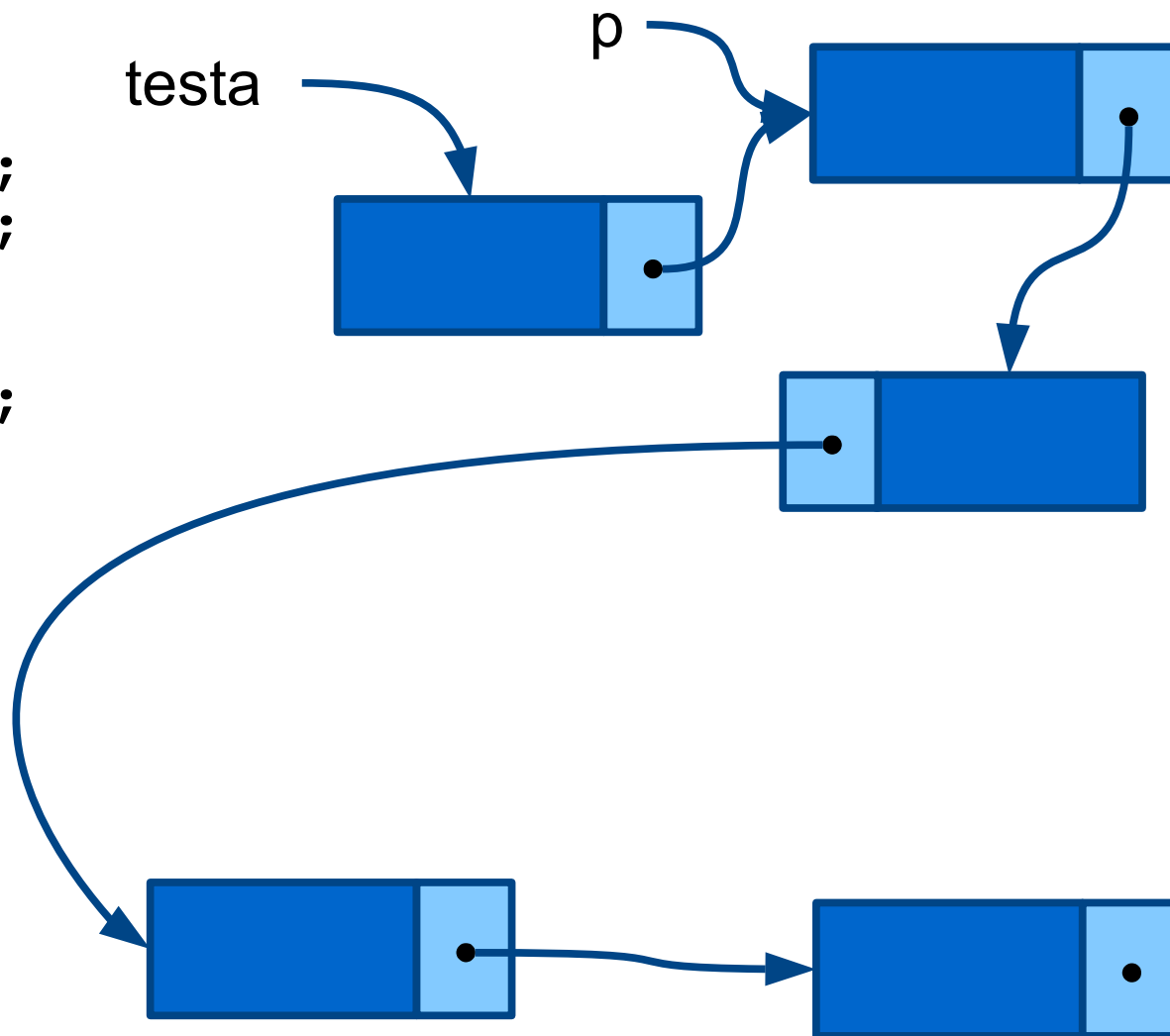
```
typedef struct Nodo
{
    int          info;
    struct Nodo* succ;
} TNode;
```

```
typedef TNode* PNode;
```

```
PNode testa, p;
```

```
...
```

```
p = testa;
while (p != NULL) {
    ...
    p = p->succ;
}
```



Scorrimento della Lista

```
typedef struct Nodo
{
    int          info;
    struct Nodo* succ;
} TNode;
```

```
typedef TNode* PNode;
```

```
PNode testa, p;
```

```
...
```

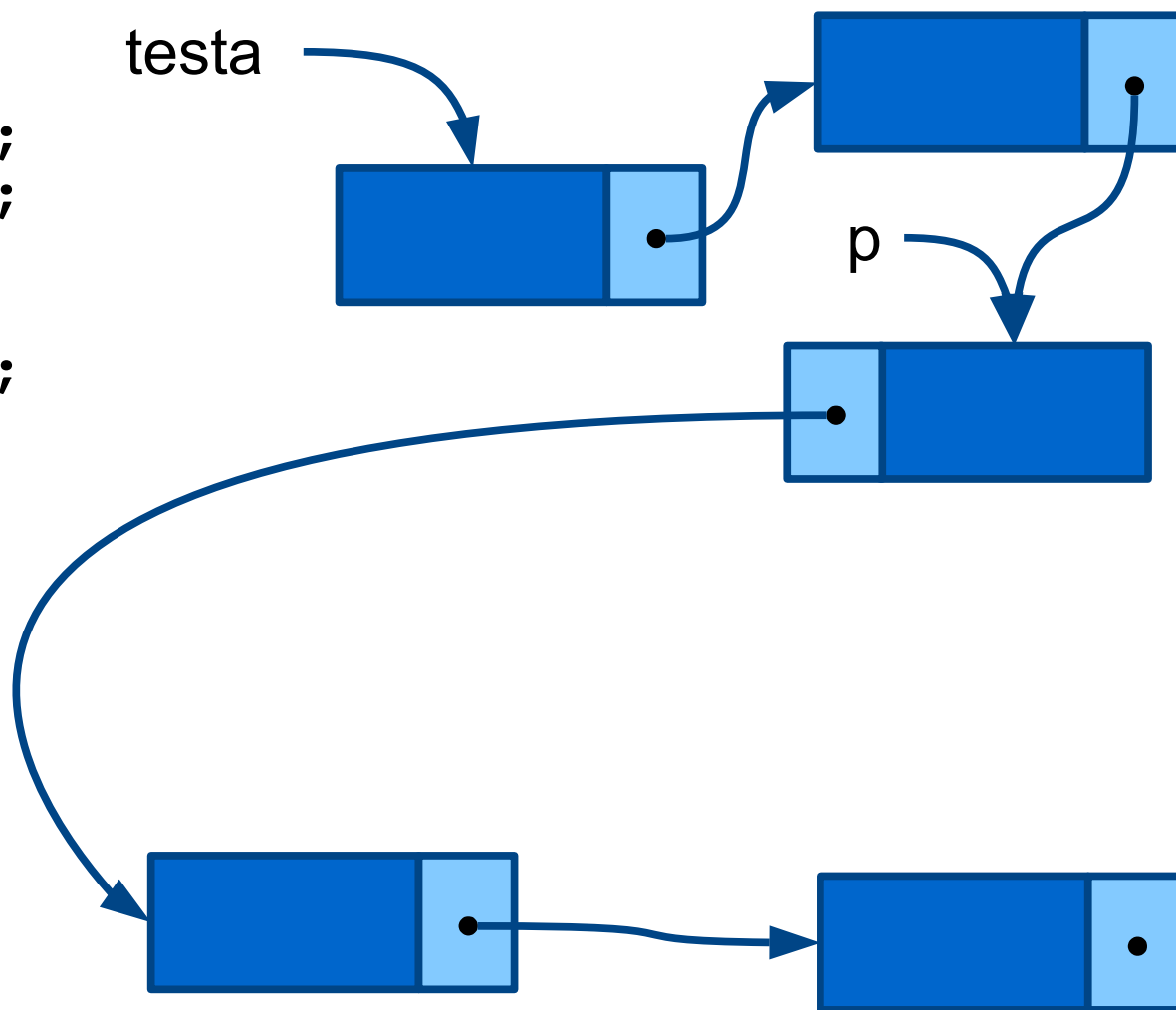
```
p = testa;
```

```
➔ while (p != NULL) {
```

```
    ...
```

```
    p = p->succ;
```

```
}
```



Scorrimento della Lista

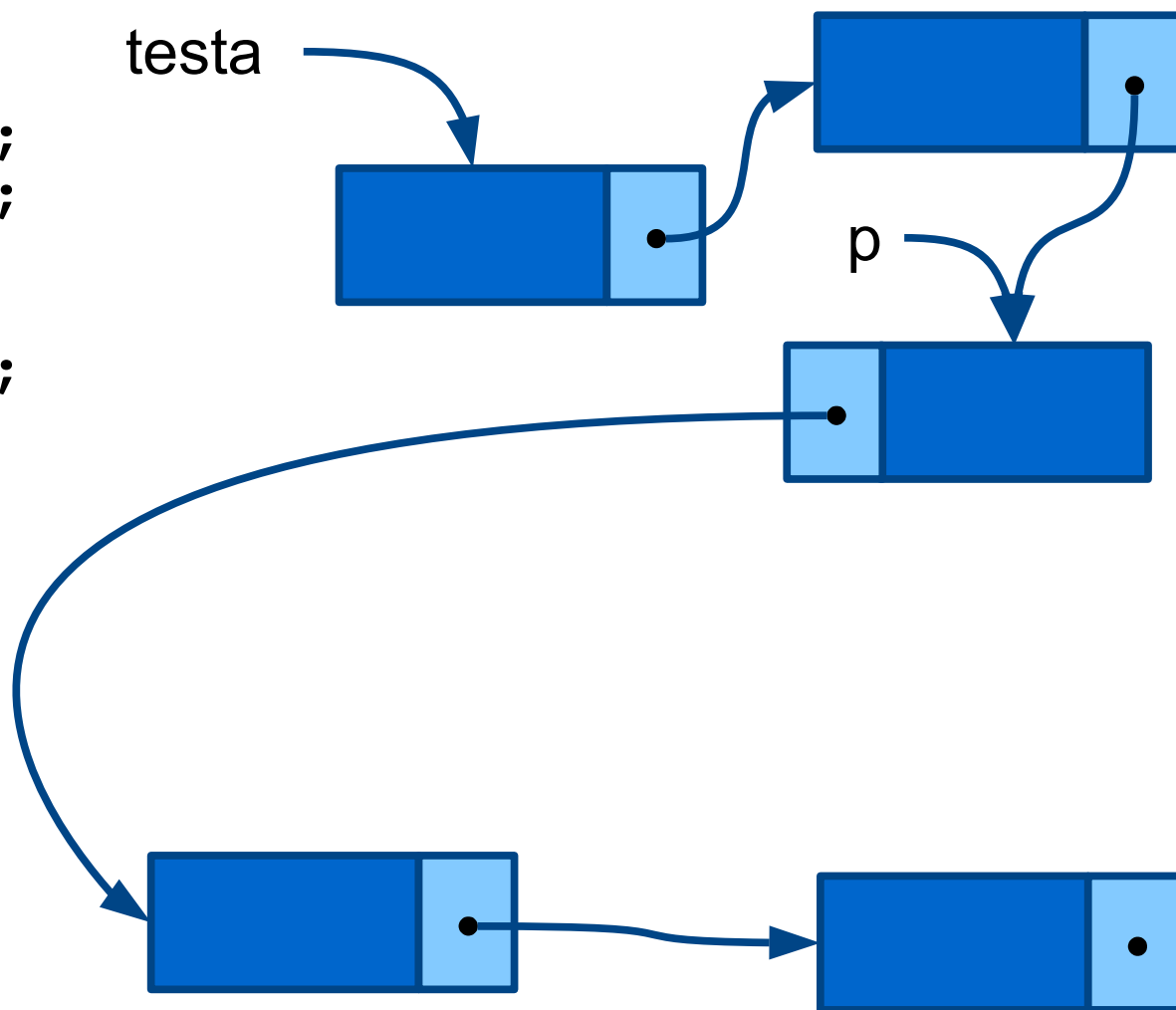
```
typedef struct Nodo
{
    int          info;
    struct Nodo* succ;
} TNode;
```

```
typedef TNode* PNode;
```

```
PNode testa, p;
```

```
...
```

```
p = testa;
while (p != NULL) {
    ...
    p = p->succ;
}
```



Scorrimento della Lista

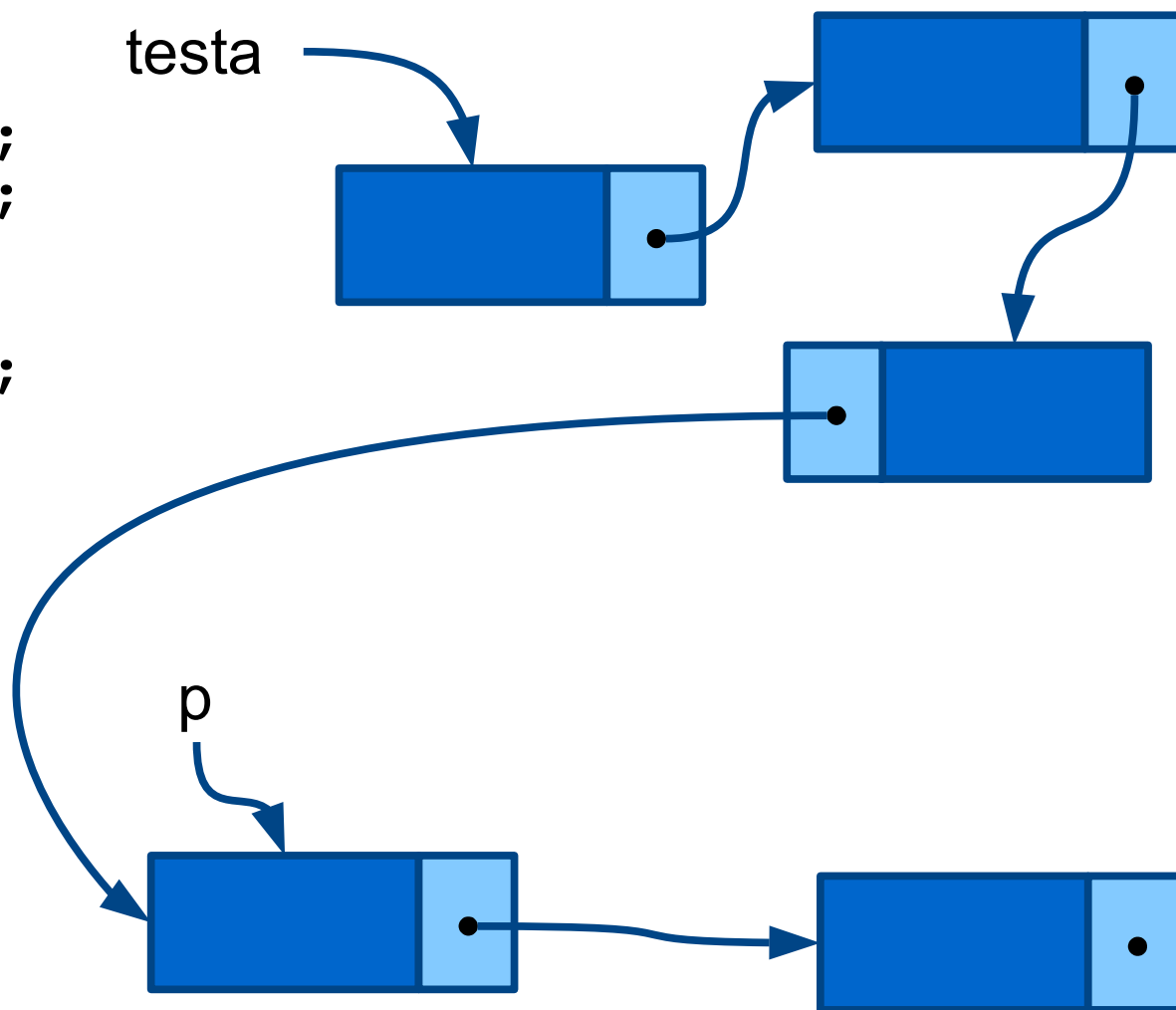
```
typedef struct Nodo
{
    int          info;
    struct Nodo* succ;
} TNode;
```

```
typedef TNode* PNode;
```

```
PNode testa, p;
```

```
...
```

```
➔ p = testa;
while (p != NULL) {
    ...
    p = p->succ;
}
```



Scorrimento della Lista

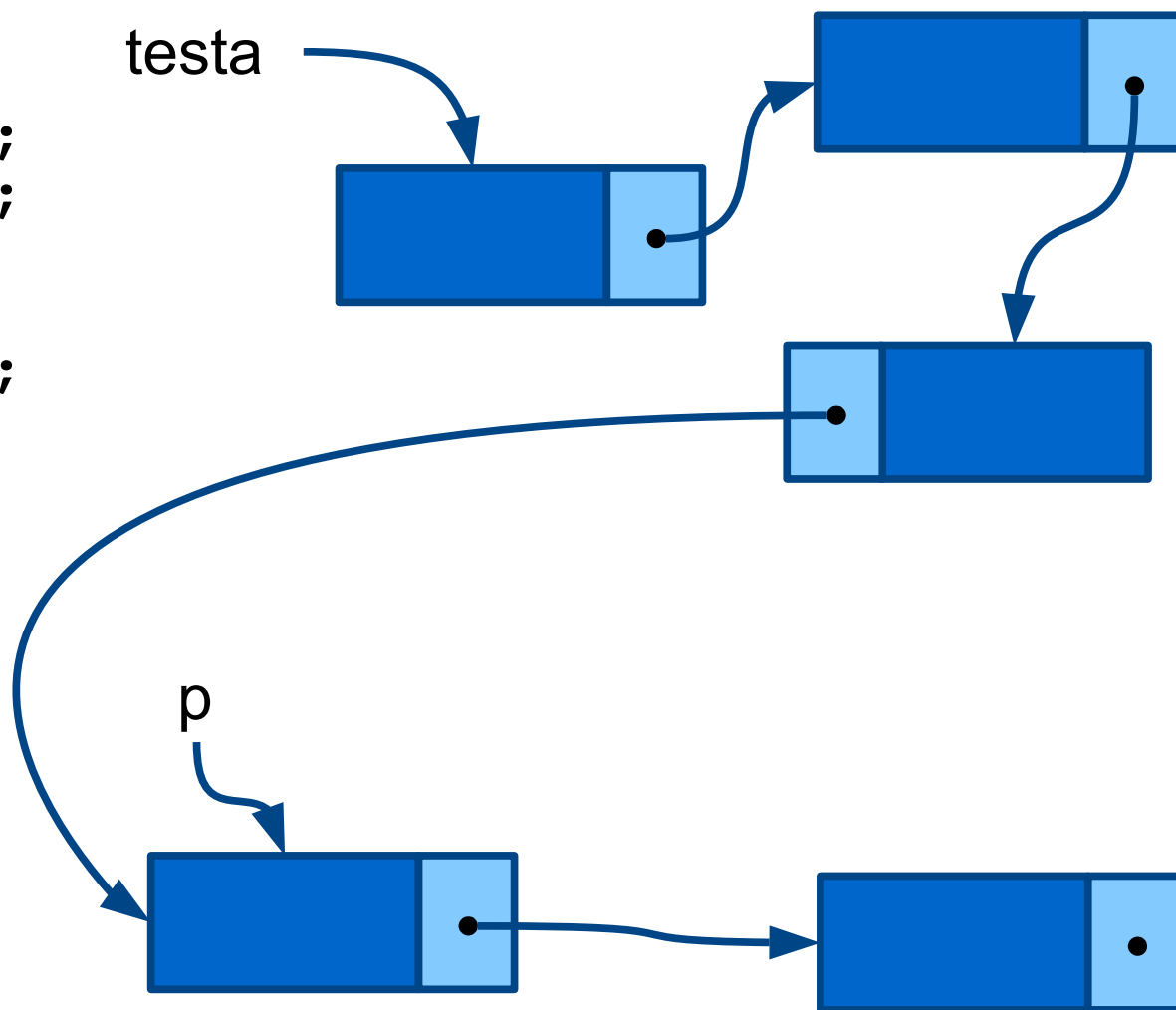
```
typedef struct Nodo
{
    int          info;
    struct Nodo* succ;
} TNode;
```

```
typedef TNode* PNode;
```

```
PNode testa, p;
```

```
...
```

```
p = testa;
while (p != NULL) {
    ...
    p = p->succ;
}
```



Scorrimento della Lista

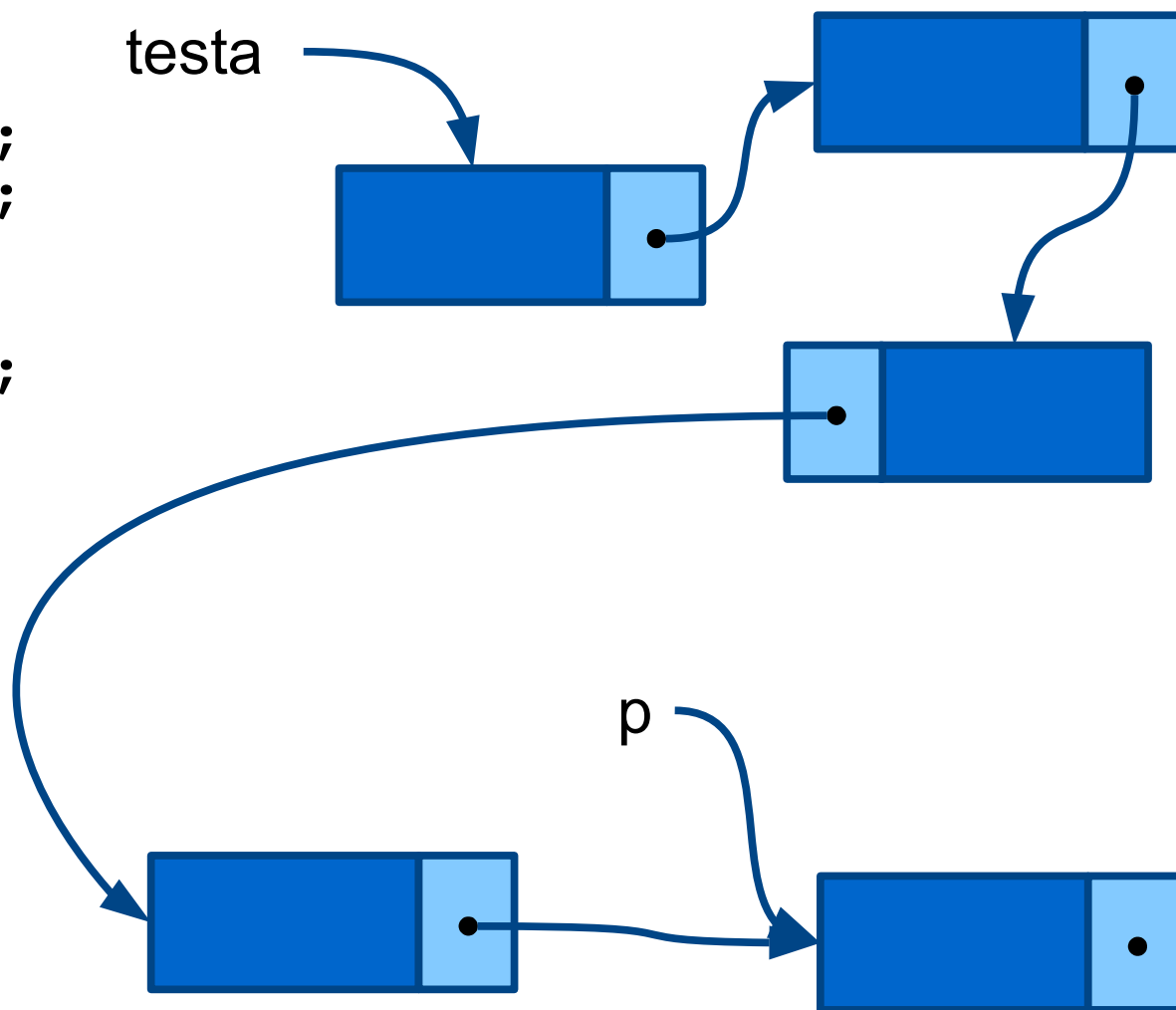
```
typedef struct Nodo
{
    int          info;
    struct Nodo* succ;
} TNode;
```

```
typedef TNode* PNode;
```

```
PNode testa, p;
```

```
...
```

```
→ p = testa;
while (p != NULL) {
    ...
    p = p->succ;
}
```



Scorrimento della Lista

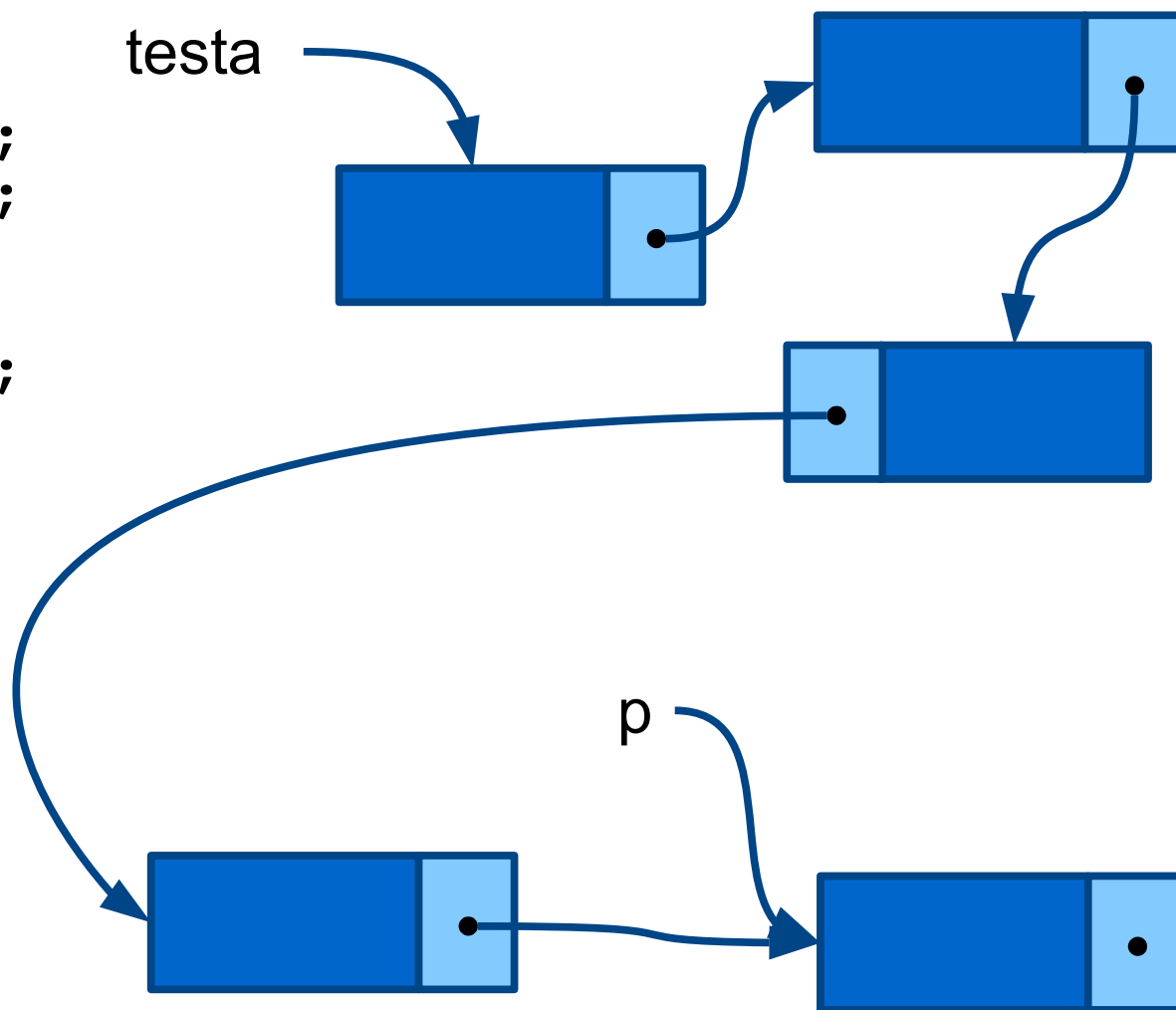
```
typedef struct Nodo
{
    int          info;
    struct Nodo* succ;
} TNode;
```

```
typedef TNode* PNode;
```

```
PNode testa, p;
```

```
...
```

```
p = testa;
while (p != NULL) {
    ...
    p = p->succ;
}
```



Scorrimento della Lista

```
typedef struct Nodo
{
    int          info;
    struct Nodo* succ;
} TNode;
```

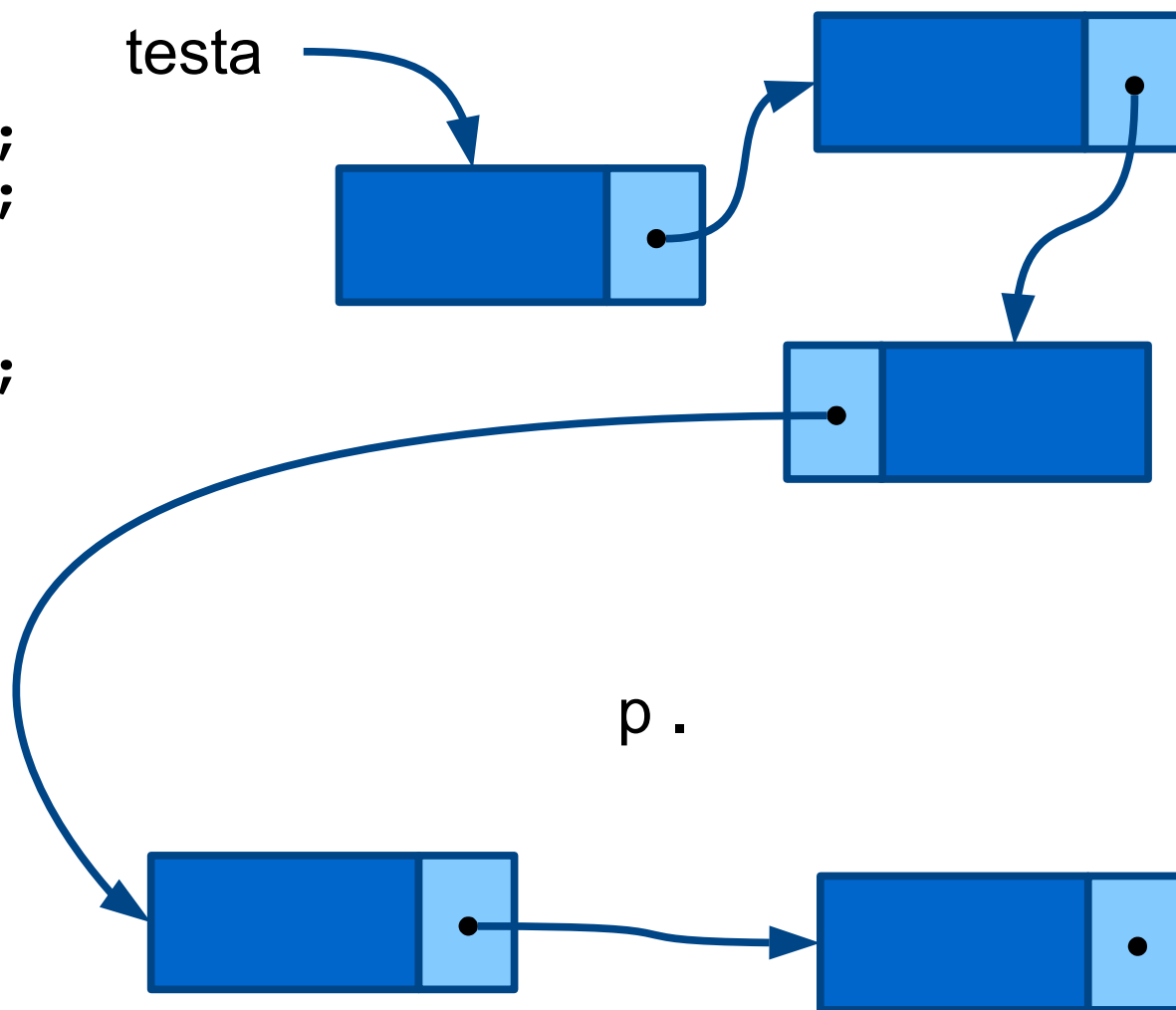
```
typedef TNode* PNode;
```

```
PNode testa, p;
```

```
...
```

```
p = testa;
```

```
→ while (p != NULL) {
    ...
    p = p->succ;
}
```



Scorrimento della Lista

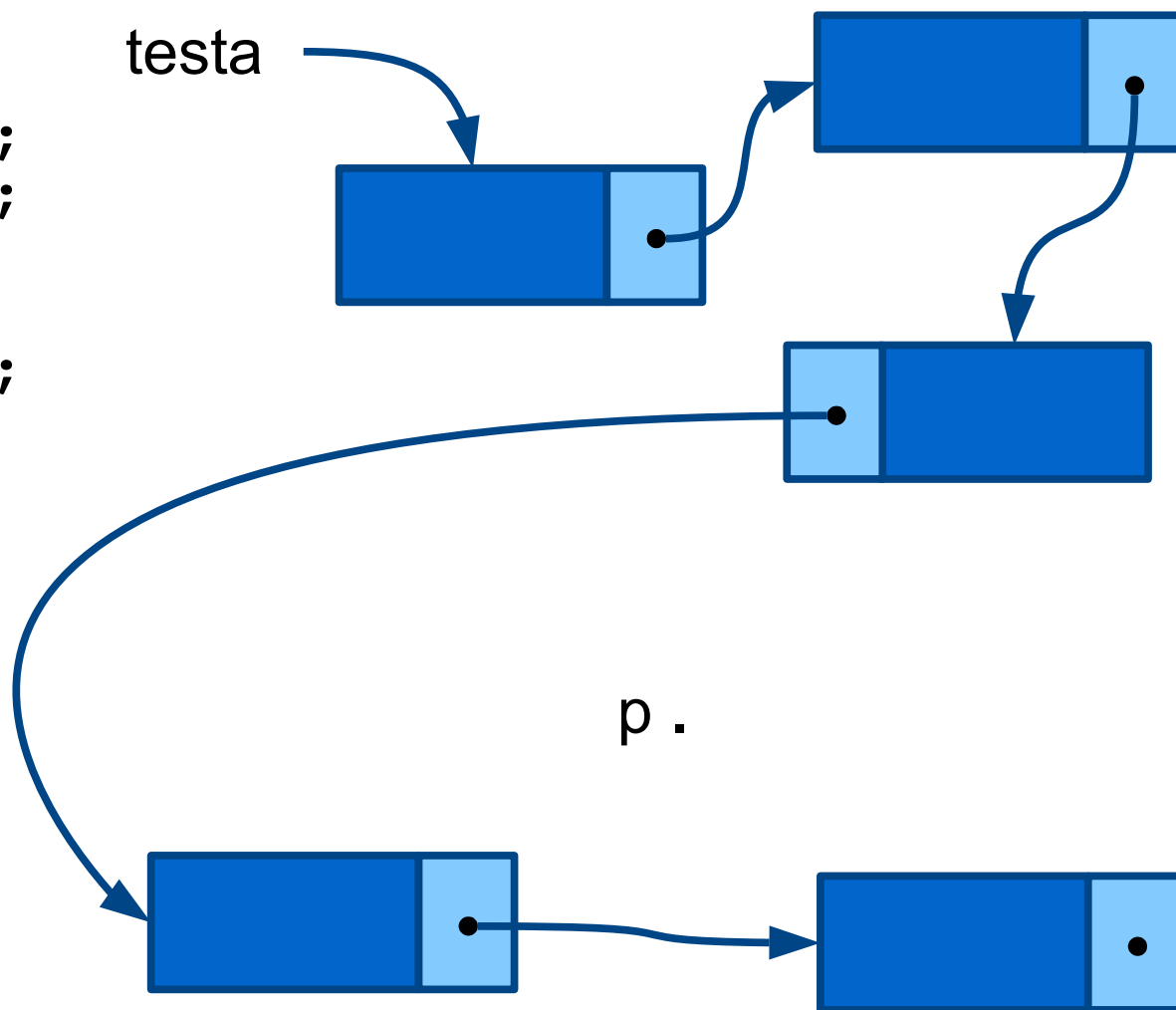
```
typedef struct Nodo
{
    int          info;
    struct Nodo* succ;
} TNode;
```

```
typedef TNode* PNode;
```

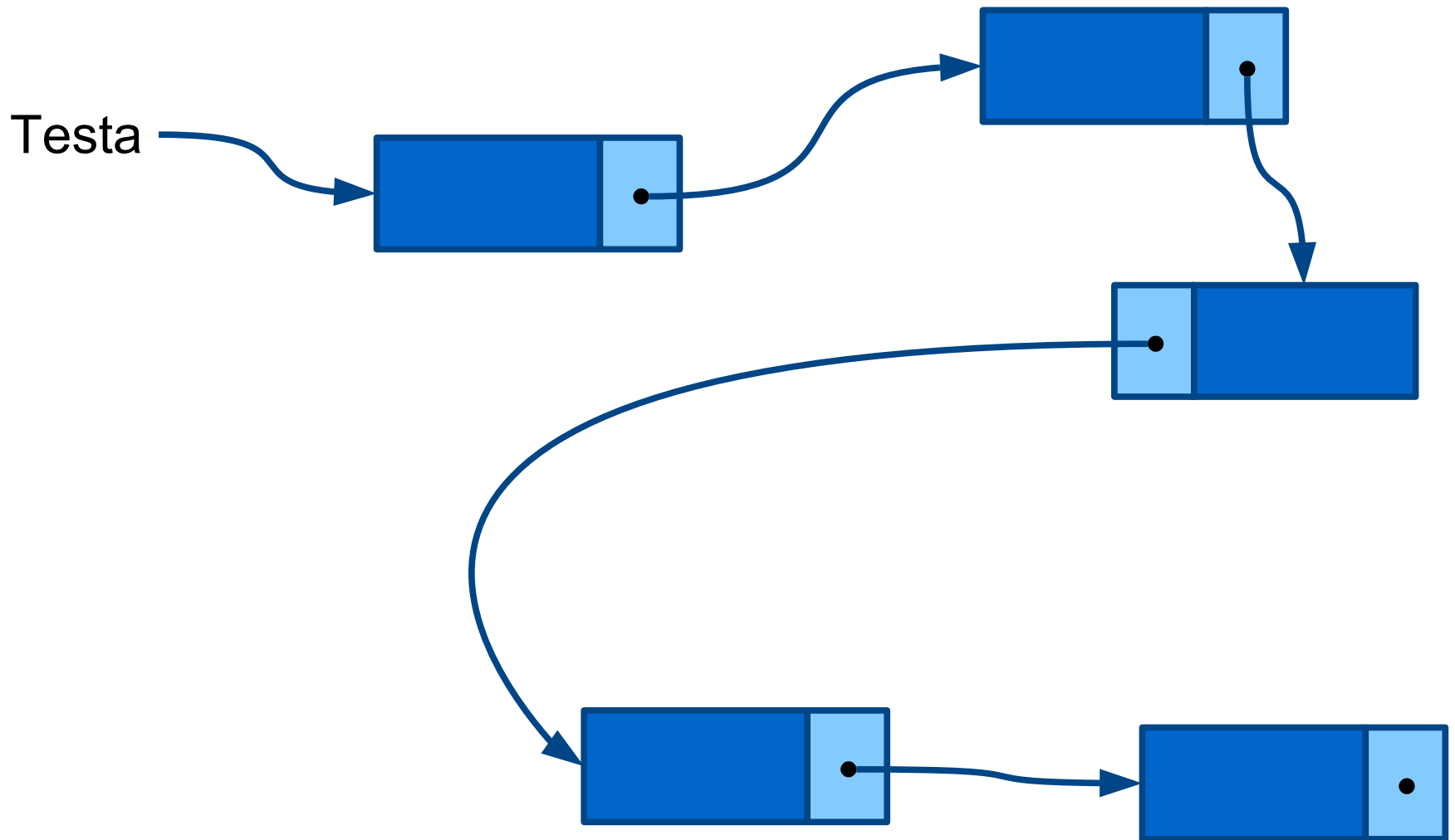
```
PNode testa, p;
```

```
...
```

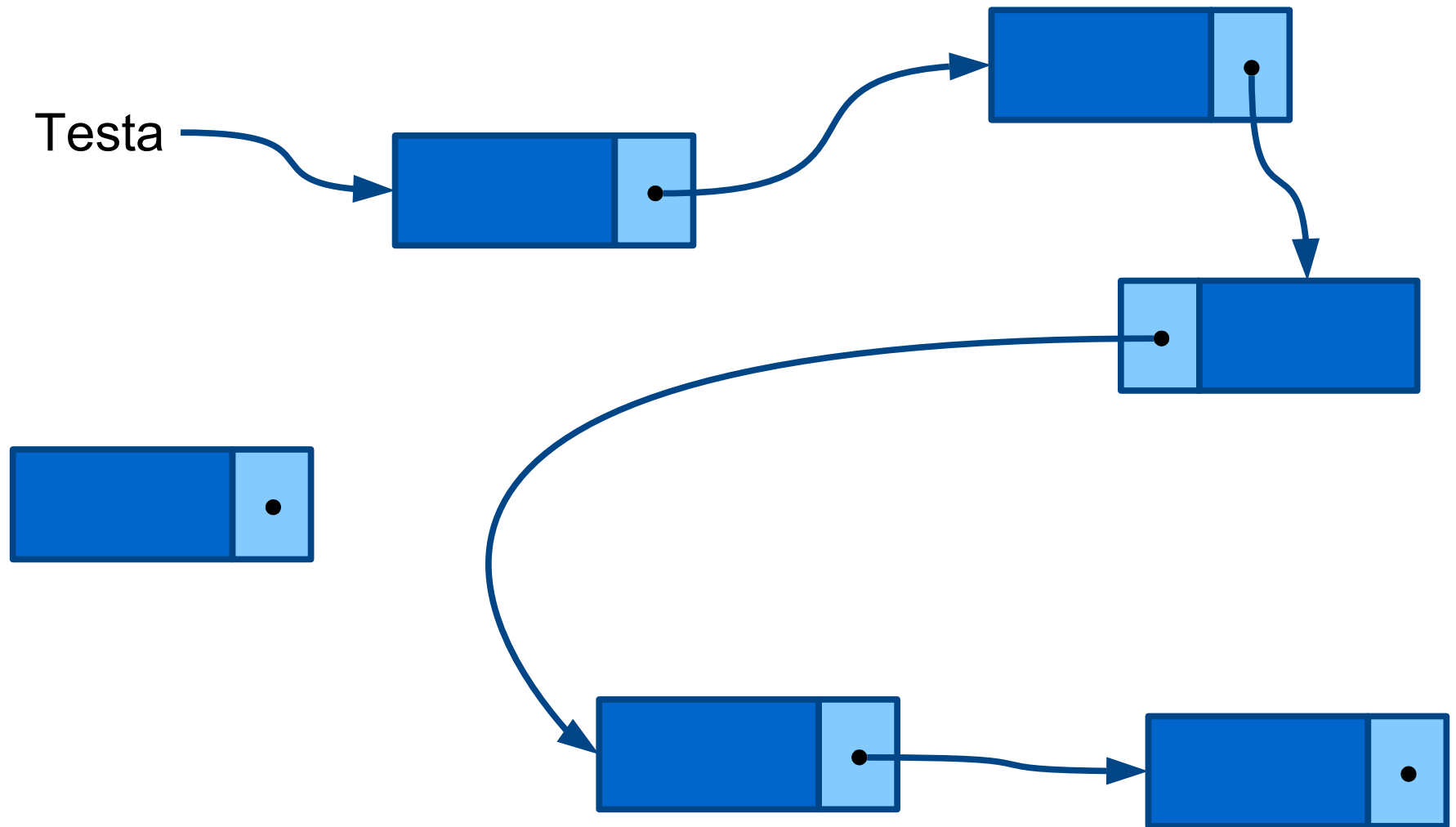
```
p = testa;
while (p != NULL) {
    ...
    p = p->succ;
}
```



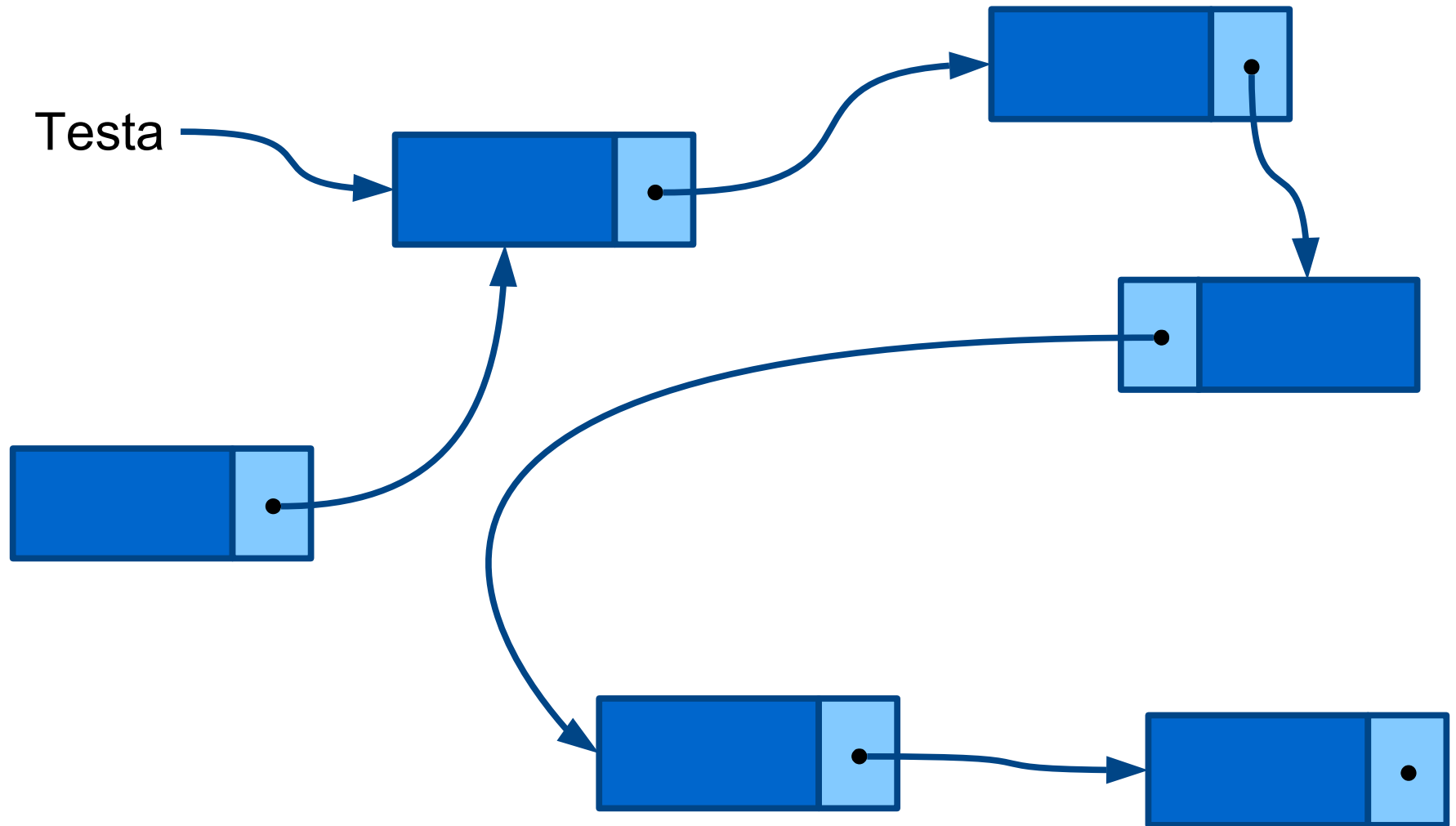
Inserimento in Testa



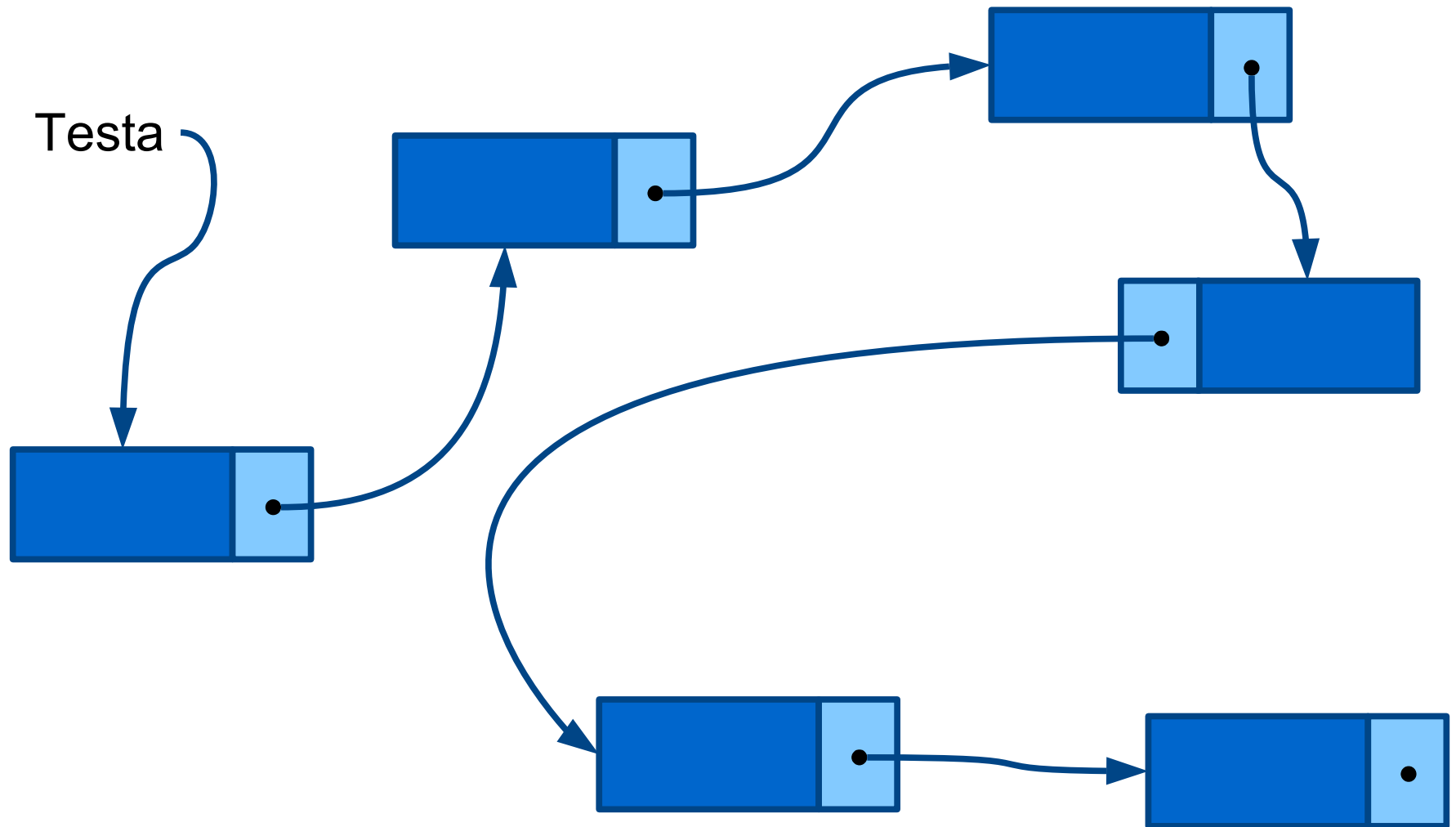
Inserimento in Testa



Inserimento in Testa



Inserimento in Testa



Inserimento in Testa

```
// Inserisce 'dato' testa alla lista e restituisce  
//il puntatore alla nuova testa  
PNodo InserisciInTesta(PNodo testa, int dato)  
{  
    PNodo pnode;  
  
    // Alloca un TNode  
    pnode = (PNodo)malloc(sizeof(TNode));  
  
    // Inizializza il nuovo nodo  
    pnode->info = dato;  
    pnode->succ = testa;  
  
    // Restituisce la nuova testa della lista  
    return pnode;  
}
```

Inserimento in Testa

```
// Inserisce 'dato' testa alla lista e restituisce  
//il puntatore alla nuova testa  
int main()  
{  
    PNode testa = NULL;  
  
    testa = InserisciInTesta(testa, 4);  
    testa = InserisciInTesta(testa, 3);  
    testa = InserisciInTesta(testa, 2);  
    testa = InserisciInTesta(testa, 1);  
  
    return 0;  
}
```

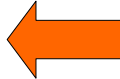
Inserimento in Testa

*// Inserisce 'dato' testa alla lista e restituisce
//il puntatore alla nuova testa*

```
int main()
```

```
{
```

```
    PNode testa = NULL;
```



```
    testa = InserisciInTesta(testa, 4);
```

```
    testa = InserisciInTesta(testa, 3);
```

```
    testa = InserisciInTesta(testa, 2);
```

```
    testa = InserisciInTesta(testa, 1);
```

```
    return 0;
```

```
}
```

testa .

Inserimento in Testa

*// Inserisce 'dato' testa alla lista e restituisce
//il puntatore alla nuova testa*

```
int main()
```

```
{
```

```
    PNode testa = NULL;
```

```
    testa = InserisciInTesta(testa, 4);
```

```
    testa = InserisciInTesta(testa, 3);
```

```
    testa = InserisciInTesta(testa, 2);
```

```
    testa = InserisciInTesta(testa, 1);
```

```
    return 0;
```

```
}
```



Inserimento in Testa

*// Inserisce 'dato' testa alla lista e restituisce
//il puntatore alla nuova testa*

```
int main()
```

```
{
```

```
    PNode testa = NULL;
```

```
    testa = InserisciInTesta(testa, 4);
```

```
    testa = InserisciInTesta(testa, 3);
```

```
    testa = InserisciInTesta(testa, 2);
```

```
    testa = InserisciInTesta(testa, 1);
```

```
    return 0;
```

```
}
```



Inserimento in Testa

*// Inserisce 'dato' testa alla lista e restituisce
//il puntatore alla nuova testa*

```
int main()
```

```
{
```

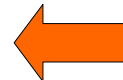
```
    PNode testa = NULL;
```

```
    testa = InserisciInTesta(testa, 4);
```

```
    testa = InserisciInTesta(testa, 3);
```

```
    testa = InserisciInTesta(testa, 2);
```

```
    testa = InserisciInTesta(testa, 1);
```



```
    return 0;
```

```
}
```



Inserimento in Testa

*// Inserisce 'dato' testa alla lista e restituisce
//il puntatore alla nuova testa*

```
int main()
```

```
{
```

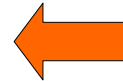
```
    PNode testa = NULL;
```

```
    testa = InserisciInTesta(testa, 4);
```

```
    testa = InserisciInTesta(testa, 3);
```

```
    testa = InserisciInTesta(testa, 2);
```

```
    testa = InserisciInTesta(testa, 1);
```

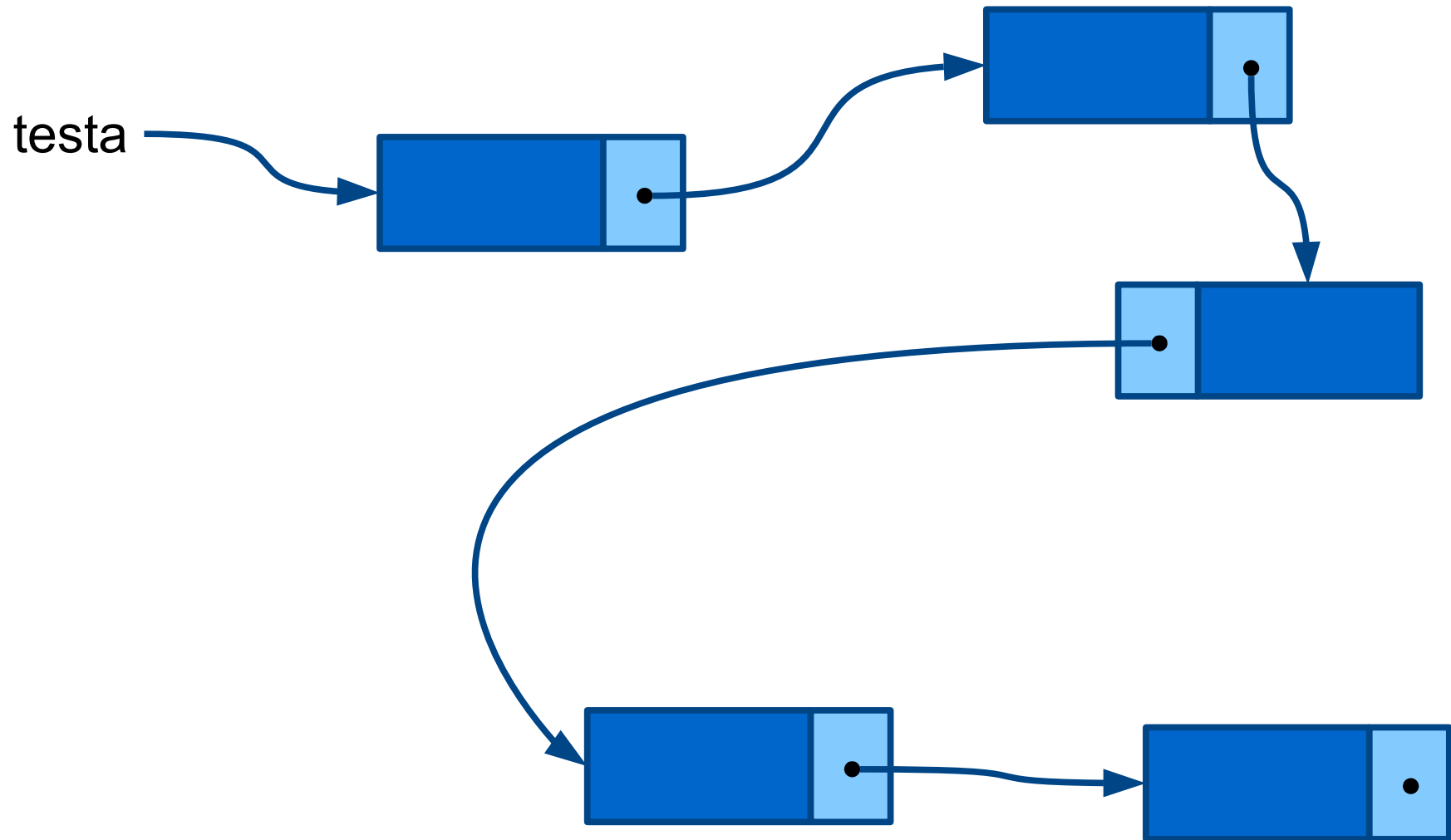


```
    return 0;
```

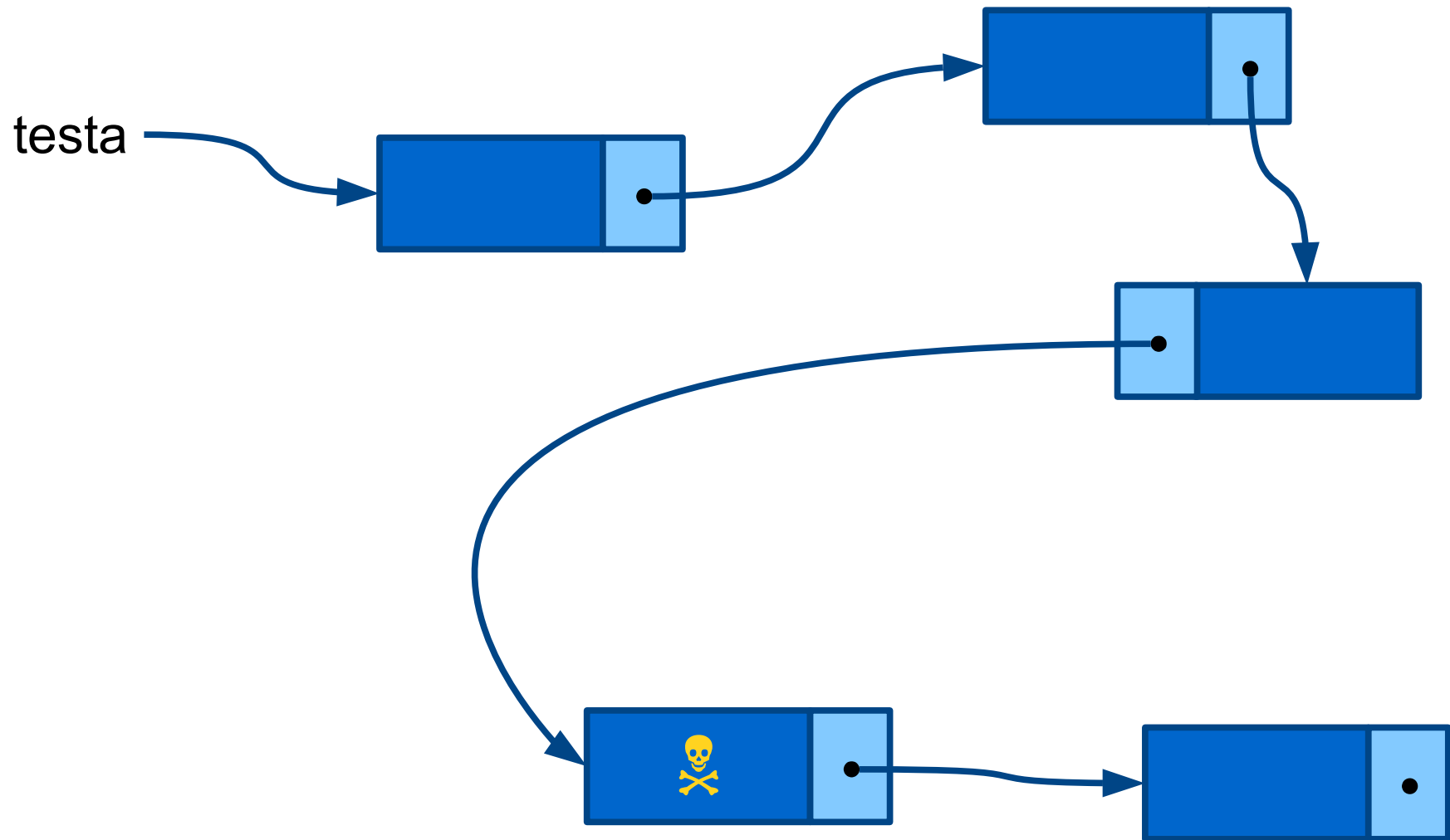
```
}
```



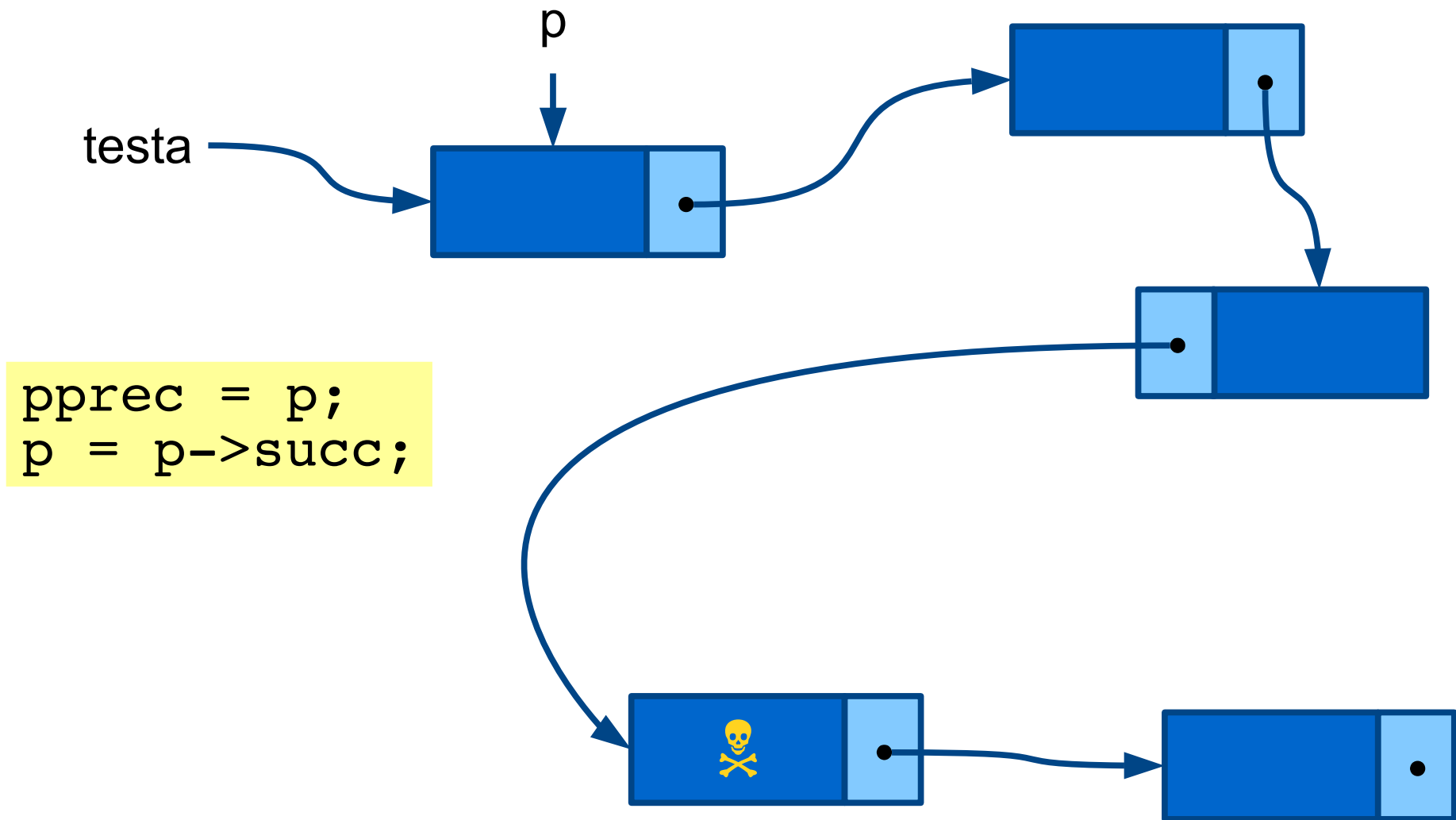
Eliminazione di un Elemento



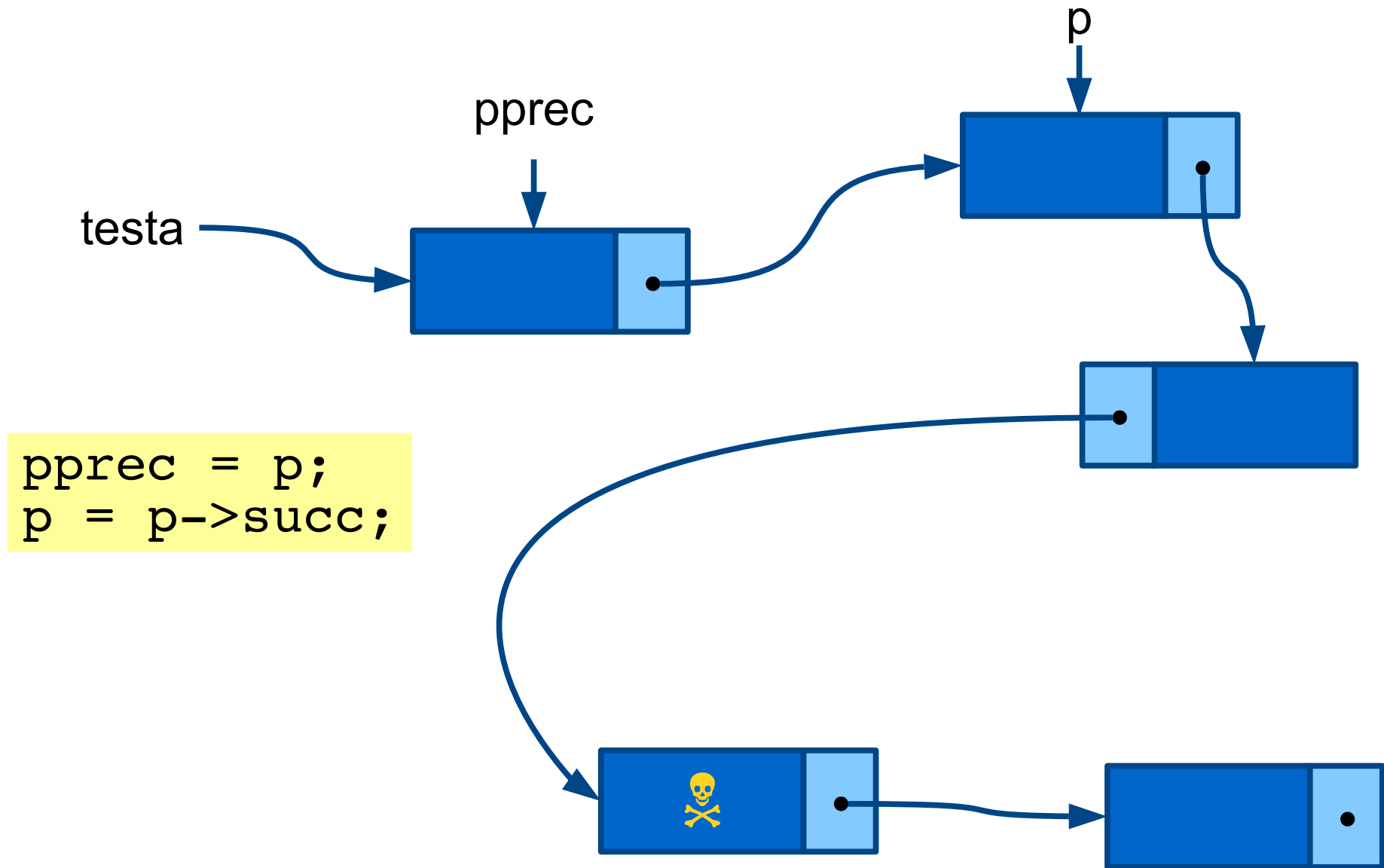
Eliminazione di un Elemento



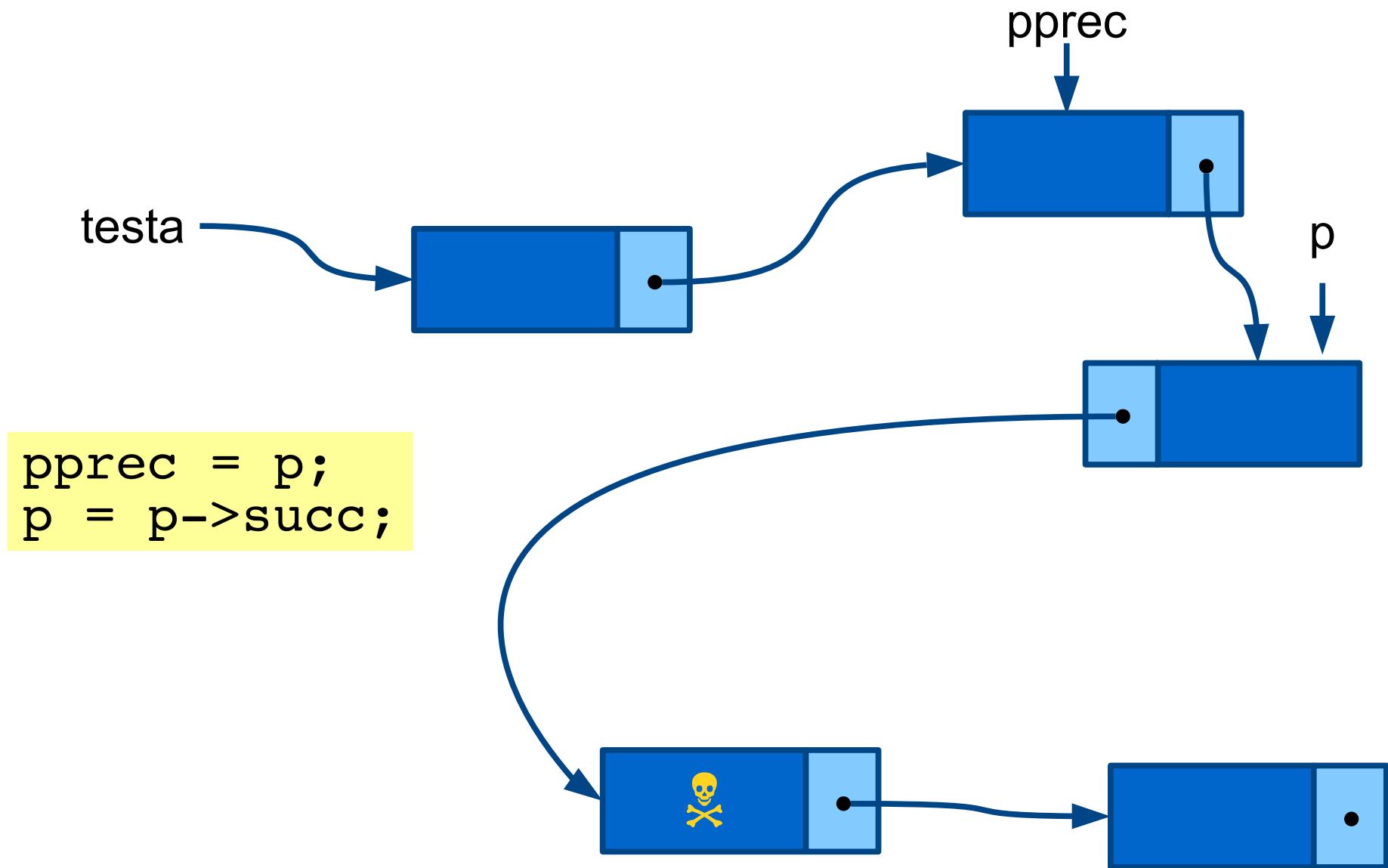
Eliminazione di un Elemento



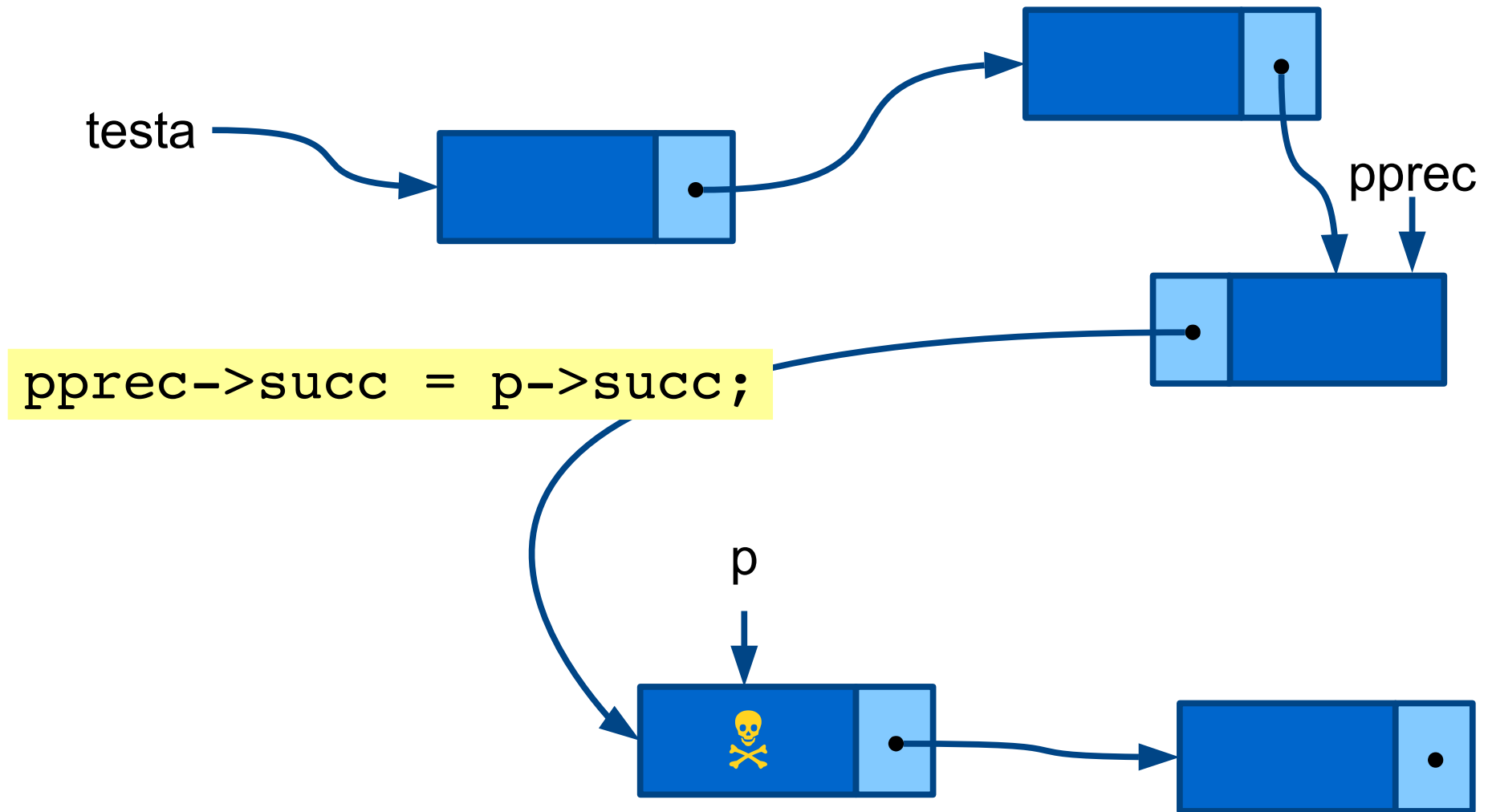
Eliminazione di un Elemento



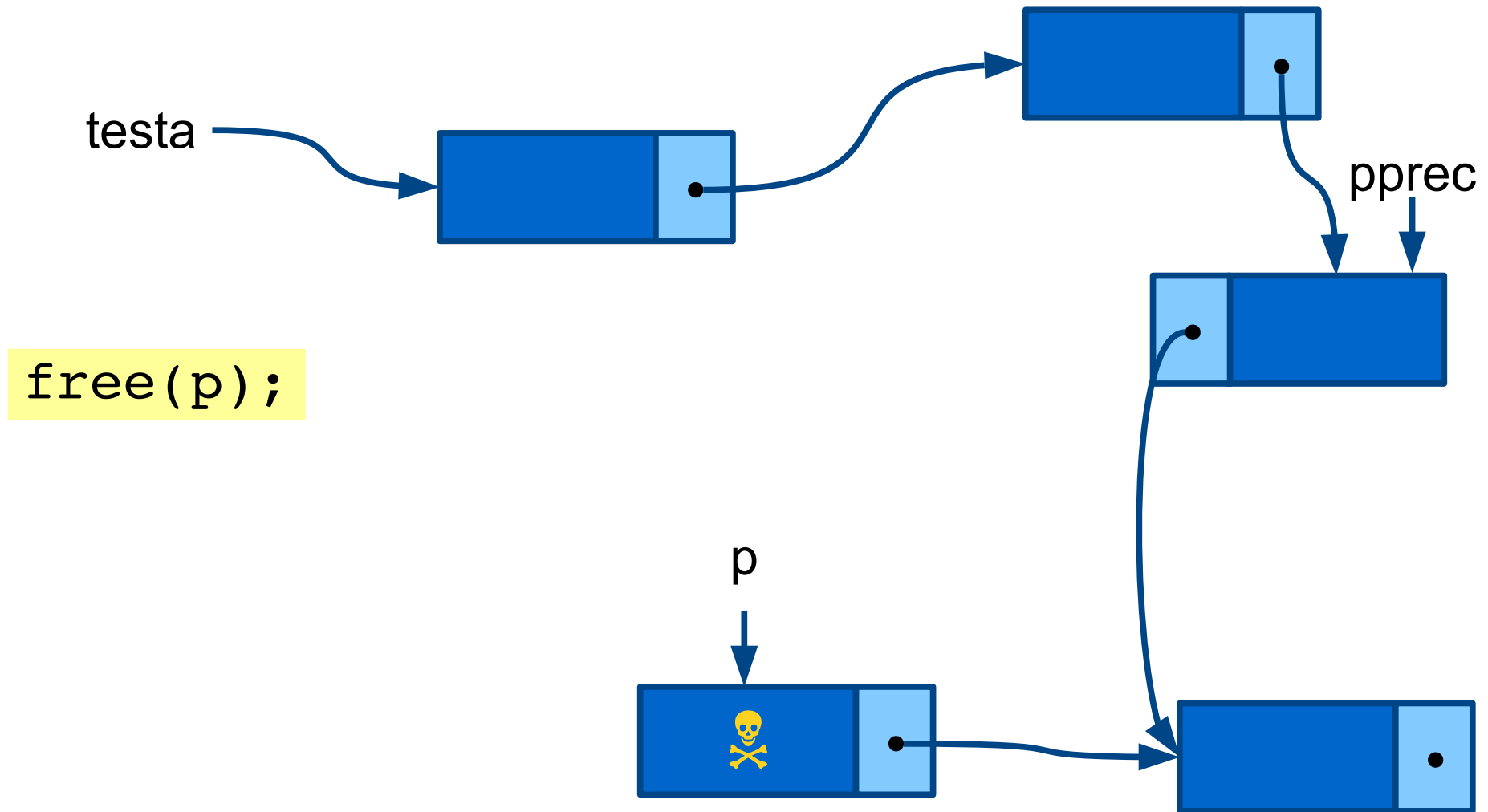
Eliminazione di un Elemento



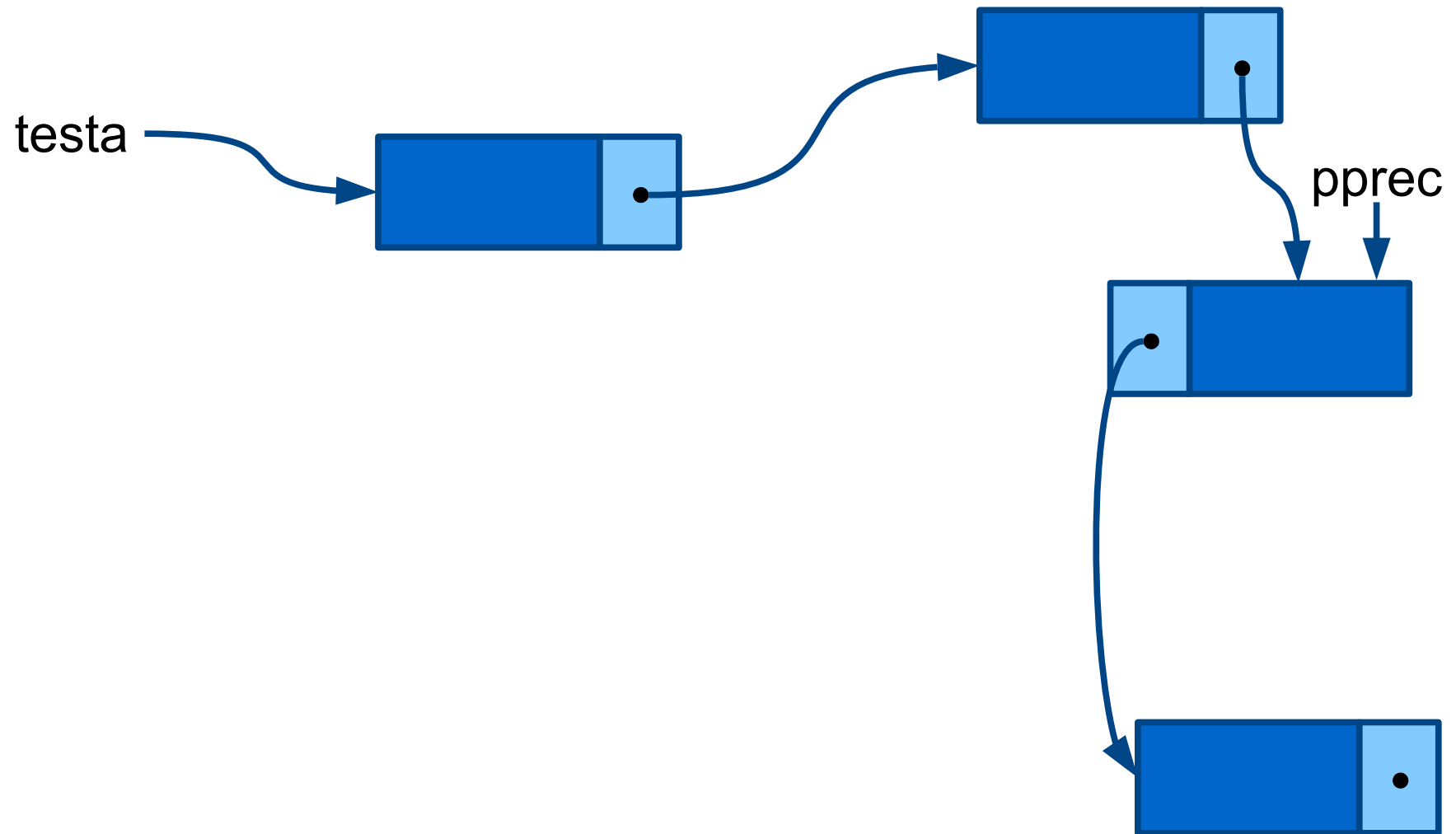
Eliminazione di un Elemento



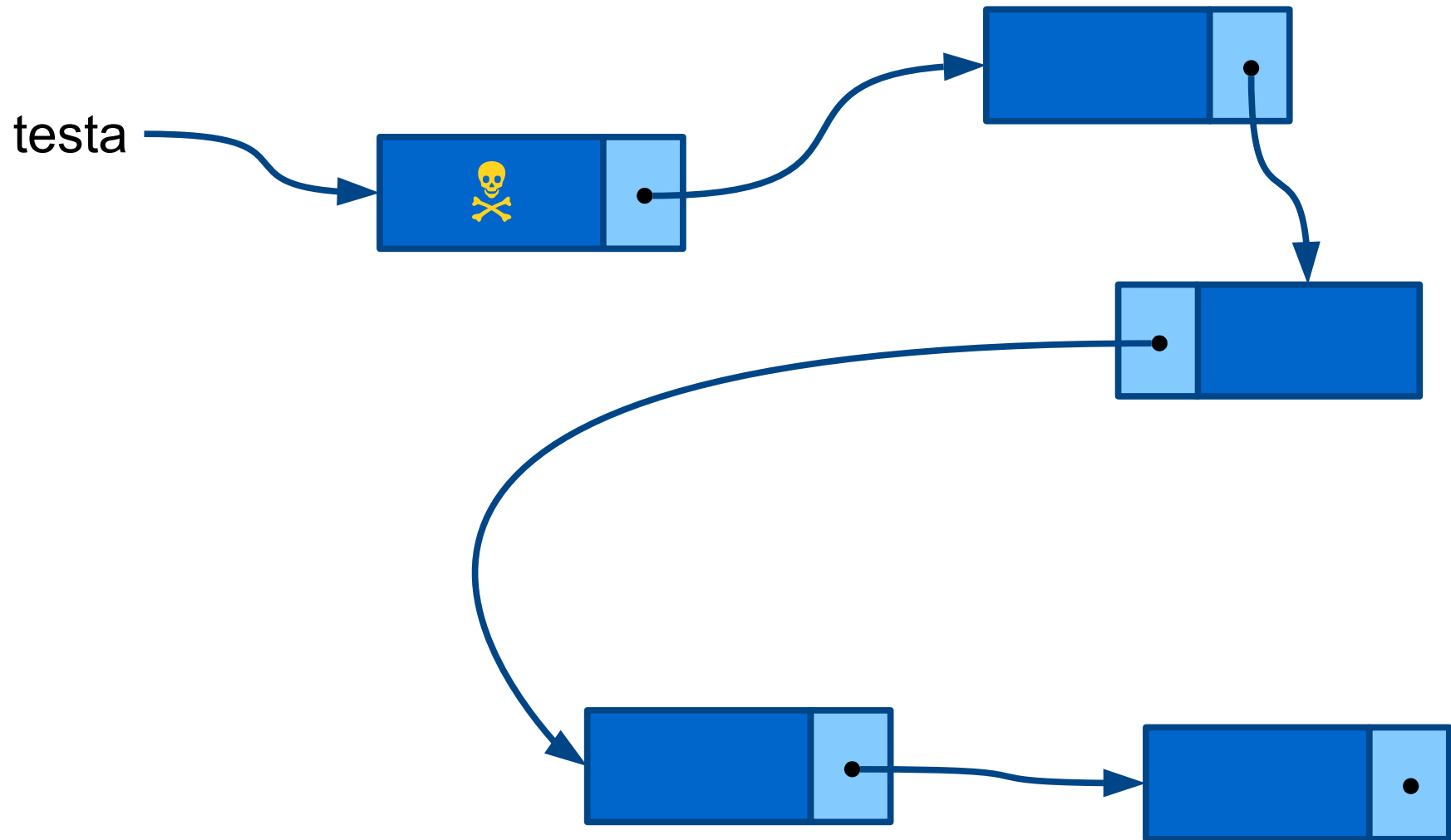
Eliminazione di un Elemento



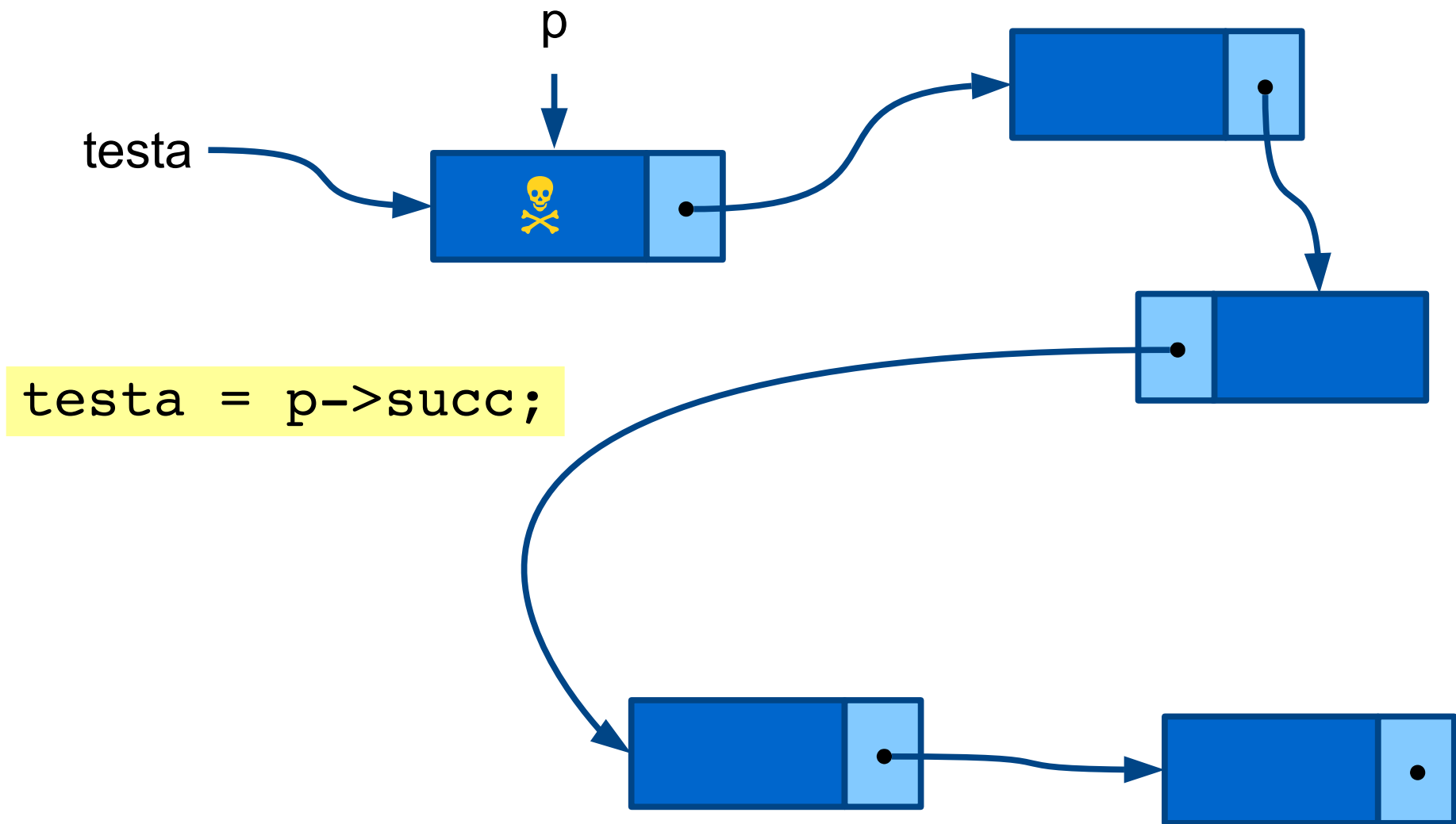
Eliminazione di un Elemento



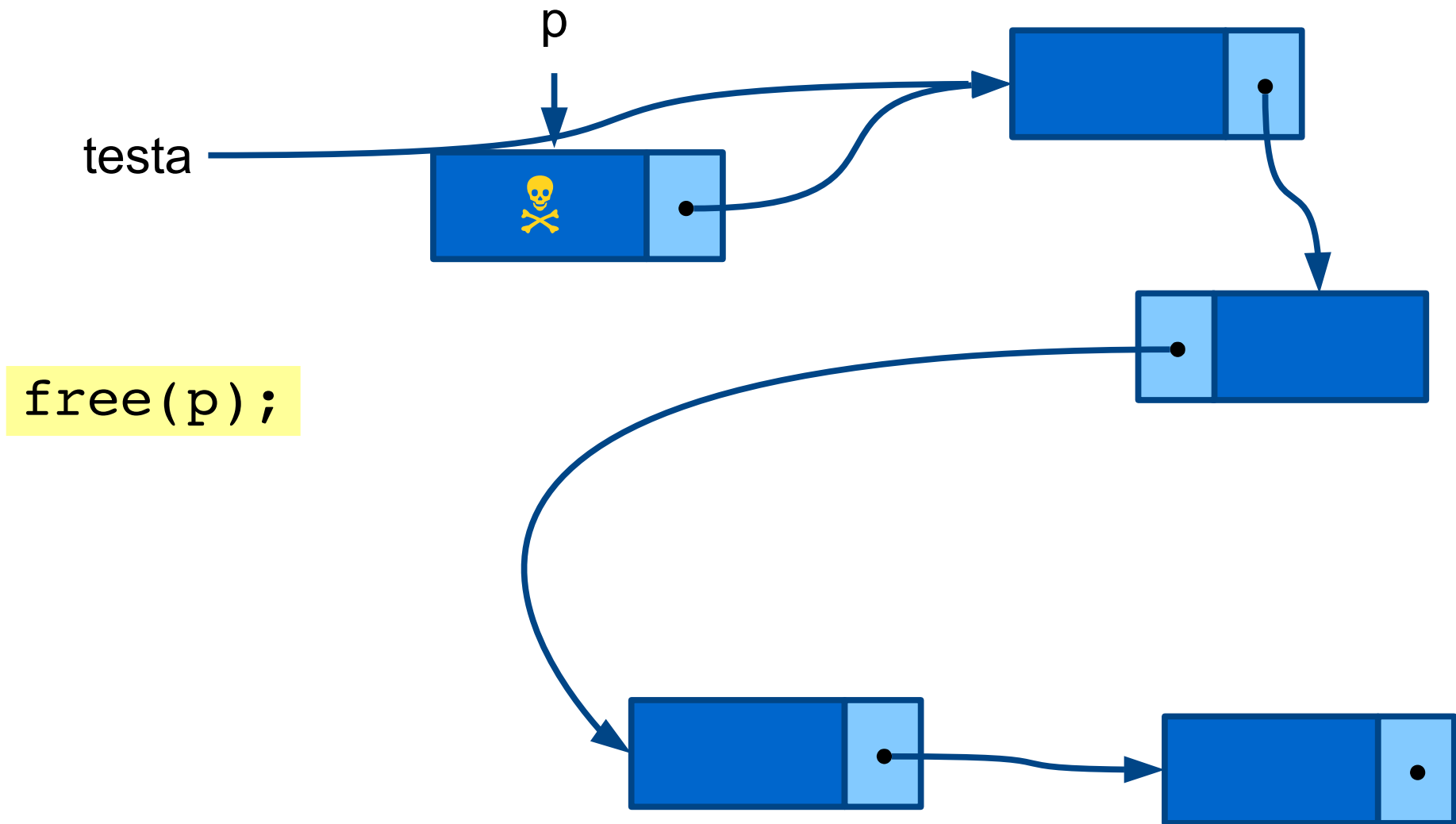
Eliminazione di un Elemento



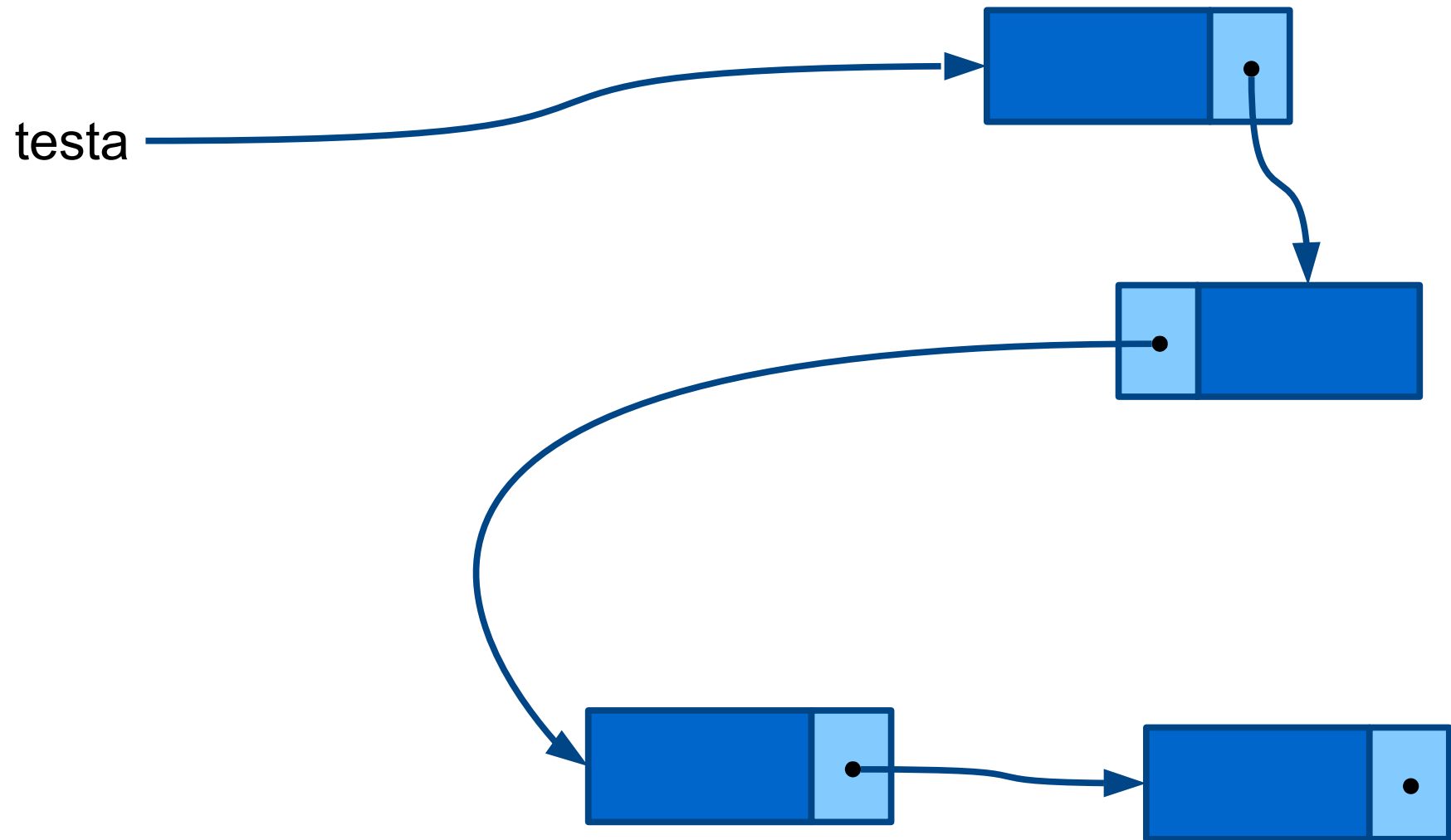
Eliminazione di un Elemento



Eliminazione di un Elemento



Eliminazione di un Elemento



Eliminazione di un Elemento

```
// Elimina il nodo della lista il cui campo info  
// corrisponde a dato. Restituisce la nuova testa  
// della lista  
PNodo EliminaNodo(PNodo testa, int dato)  
{  
    PNodo p, pprec;  
  
    pprec = NULL;  
    p = testa;  
  
    // Cerco l'elemento da eliminare  
    while (p != NULL && p->info != dato)  
    {  
        pprec = p;  
        p = p->succ;  
    }  
}
```


Eliminazione di un Elemento

```
if (p == NULL)
    return testa;

// Elimino il nodo puntato da p
if (p == testa) // Eliminazione del nodo di testa
    testa = p->succ;
else // Eliminazione di un nodo interno alla
     // lista
    pprec->succ = p->succ;

free(p);

return testa;
}
```

Classificazione

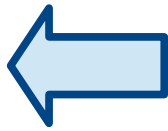
- Lista
 - Pila
 - Coda

Pila

- Struttura LIFO
 - Last Input First Output
- Due operazioni base
 - **Push**: inserisce un elemento in cima alla pila
 - **Pop**: estrae un elemento dalla cima della pila

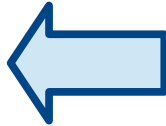
Esempio – Pila

```
Push( 'C' );  
Push( 'i' );  
Push( 'a' );  
Push( 'o' );  
c = Pop( );  
c = Pop( );  
c = Pop( );
```



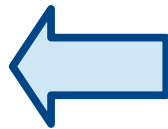
Esempio – Pila

```
Push( 'C' );  
Push( 'i' );  
Push( 'a' );  
Push( 'o' );  
c = Pop( );  
c = Pop( );  
c = Pop( );
```

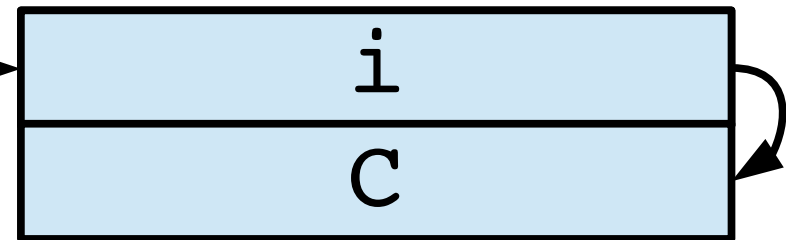


Esempio – Pila

```
Push( 'C' );  
Push( 'i' );  
Push( 'a' );  
Push( 'o' );  
c = Pop( );  
c = Pop( );  
c = Pop( );
```



t →

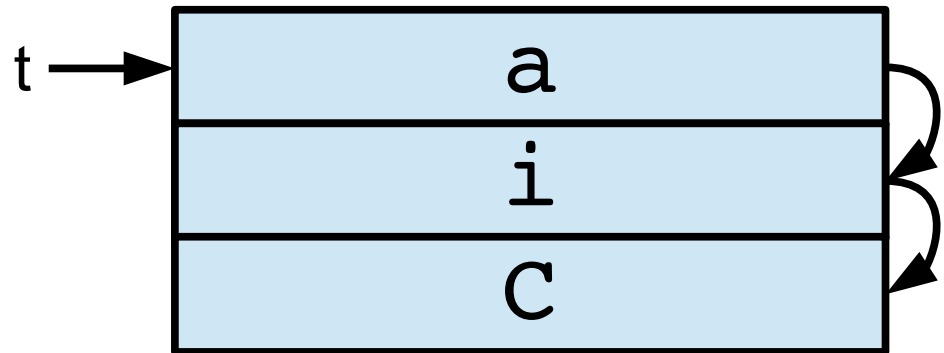
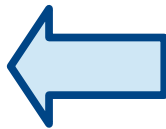


c



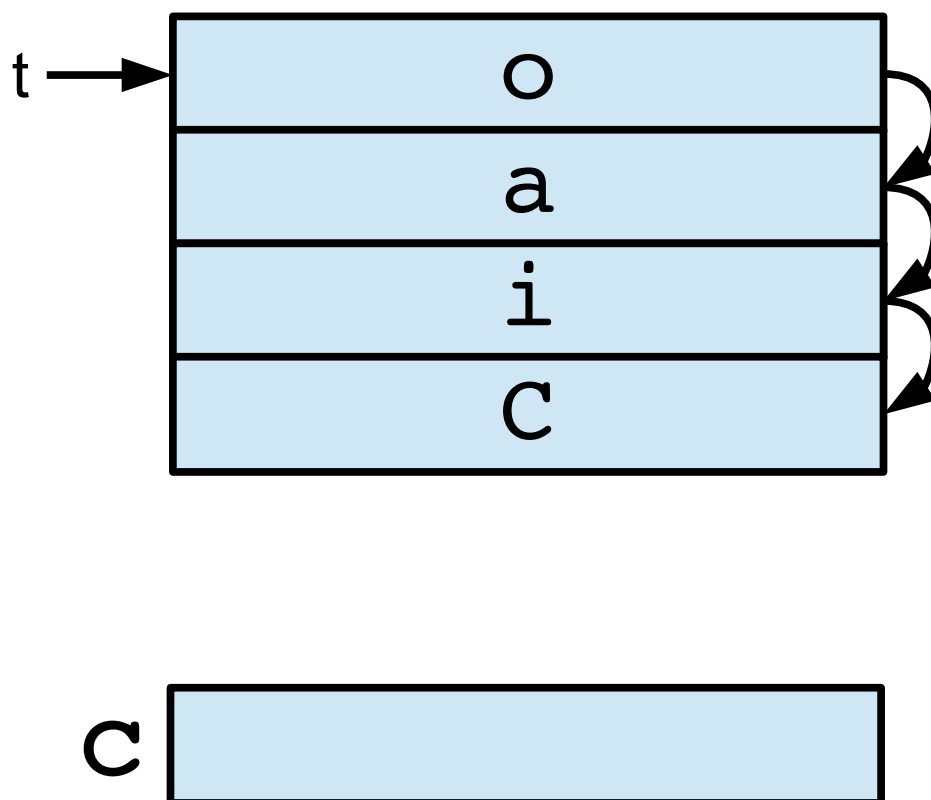
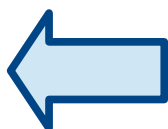
Esempio – Pila

```
Push( 'C' );  
Push( 'i' );  
Push( 'a' );  
Push( 'o' );  
c = Pop( );  
c = Pop( );  
c = Pop( );
```



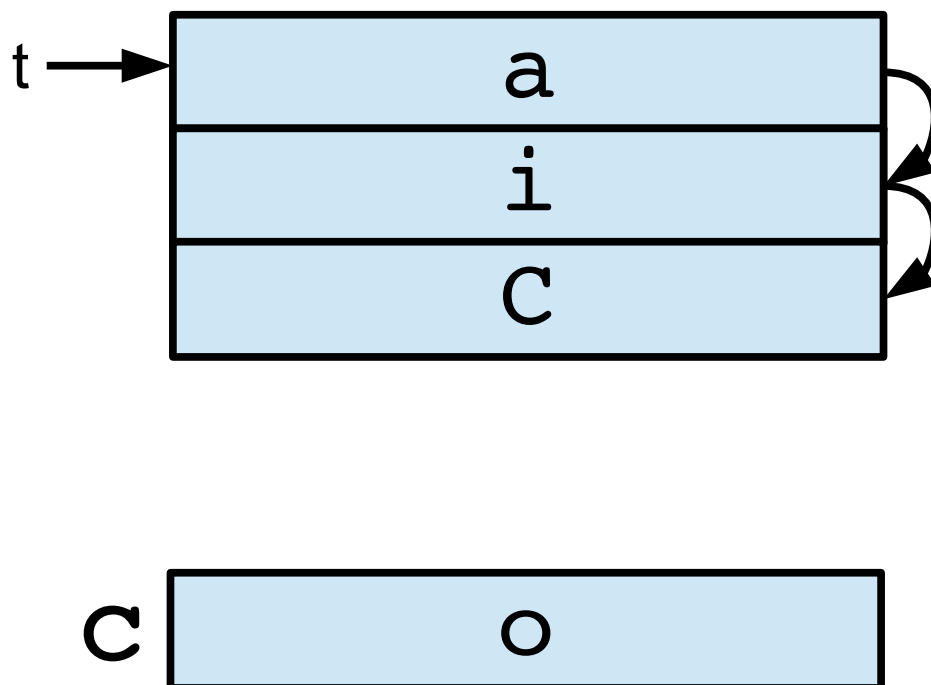
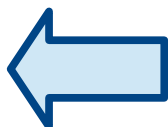
Esempio – Pila

```
Push( 'C' );  
Push( 'i' );  
Push( 'a' );  
Push( 'o' );  
c = Pop( );  
c = Pop( );  
c = Pop( );
```



Esempio – Pila

```
Push( 'C' );  
Push( 'i' );  
Push( 'a' );  
Push( 'o' );  
c = Pop( );  
c = Pop( );  
c = Pop( );
```



Esempio – Pila

```
Push( 'C' );
```

```
Push( 'i' );
```

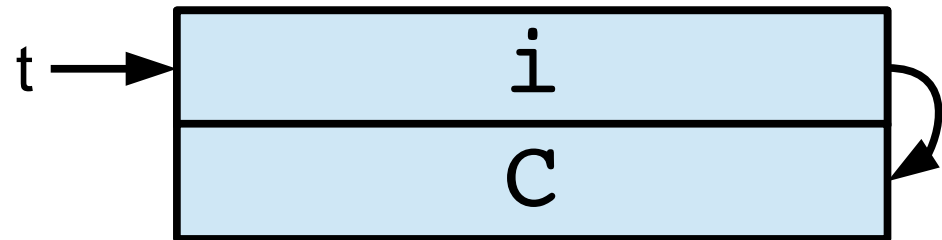
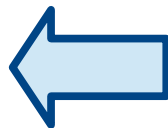
```
Push( 'a' );
```

```
Push( 'o' );
```

```
c = Pop( );
```

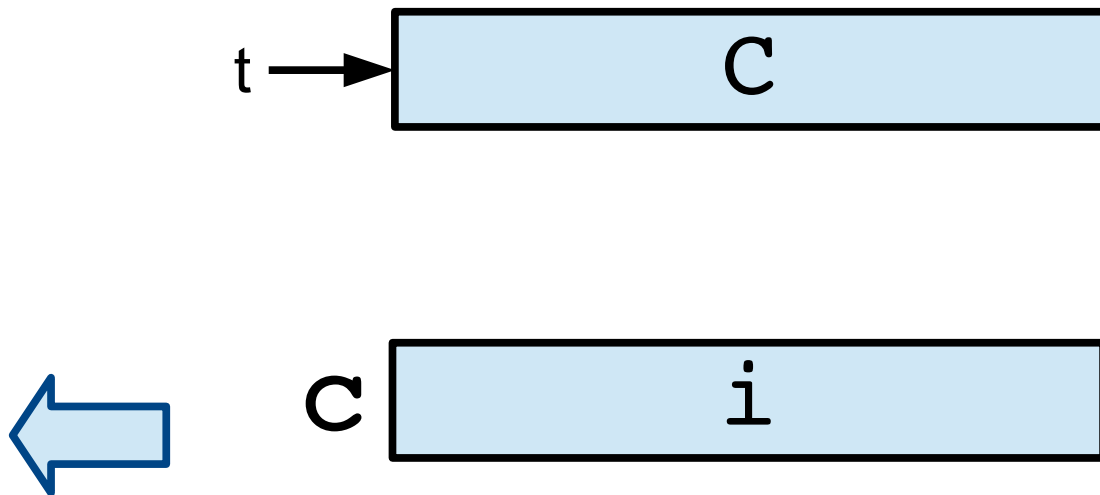
```
c = Pop( );
```

```
c = Pop( );
```



Esempio – Pila

```
Push( 'C' );  
Push( 'i' );  
Push( 'a' );  
Push( 'o' );  
c = Pop( );  
c = Pop( );  
c = Pop( );
```



Implementazione in C

```
// Struttura dati
typedef struct Nodo
{
    int          info;
    struct Nodo* succ;
} TNode;

typedef TNode* PPila;
```

Implementazione in C (*cont.*)

```
// Inserisce n in cima alla pila
void Push(PPila* pila, int n)
{
    TNode* p;

    p = (TNode*)malloc(sizeof(TNode));
    p->info = n;
    p->succ = *pila;

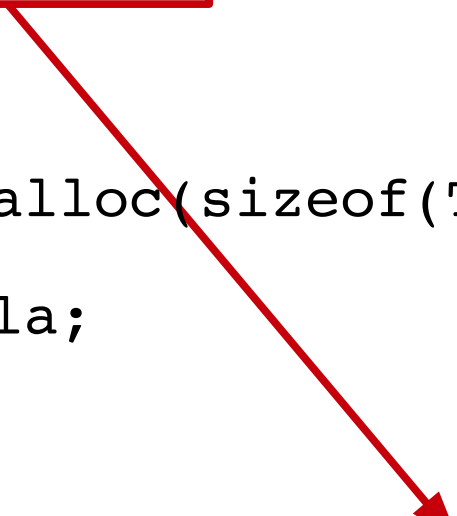
    *pila = p;
}
```

Implementazione in C (*cont.*)

```
// Inserisce n in cima alla pila
void Push(PPila* pila, int n)
{
    TNode* p;

    p = (TNode*)malloc(sizeof(TNode));
    p->info = n;
    p->succ = *pila;

    *pila = p;
}
```



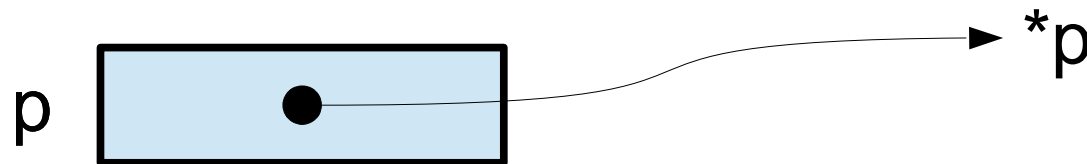
PPila è un TNode* quindi pila è un TNode** cioè un puntatore a puntatore! Si richiede il puntatore a puntatore poiché l' inserimento nella lista dovrà alterare la testa della lista

Nota

```
int* p;
```

```
p = malloc(...);  
ModificaValorePuntato(p);
```

```
void ModificaValorePuntato(int* q)  
{  
    *q = <nuovo valore>  
}
```

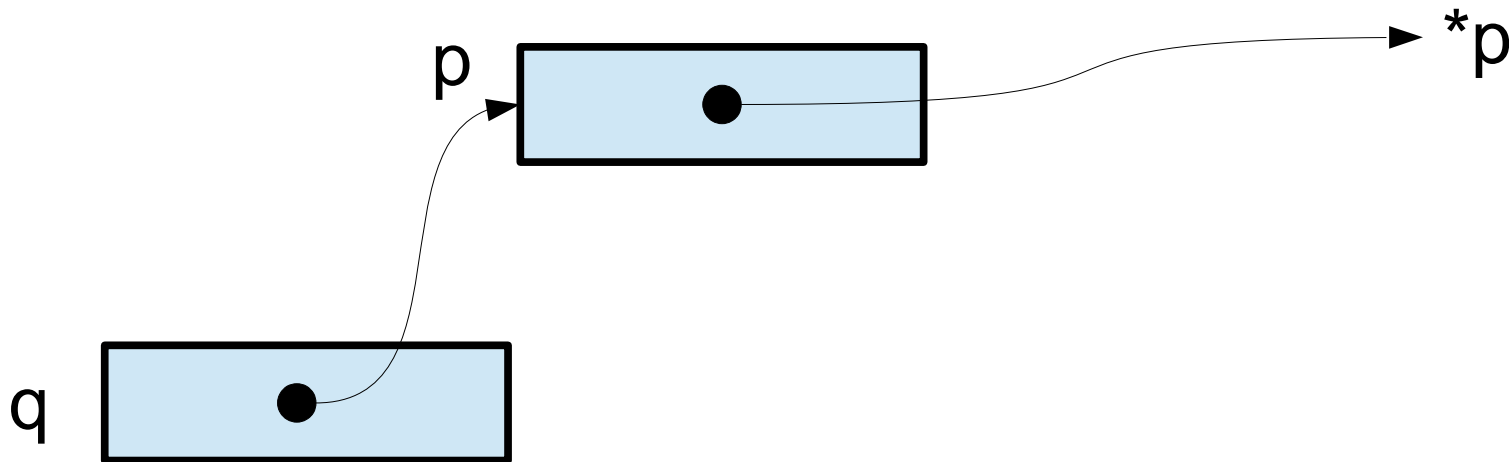


Nota

```
int* p;
```

```
p = malloc(...);  
ModificaPuntatore(&p);
```

```
void ModificaValorePuntato(int** q)  
{  
    *q = NULL;  
}
```



Implementazione in C (*cont.*)

```
// Estrae l'elemento in cima alla pila.  
// Restituisce 0 se la pila è vuota e 1 altrimenti  
int Pop(PPila* pila, int* n)  
{  
    TNode* p;  
  
    if (*pila == NULL)  
        return 0;  
  
    *n = (*pila)->info;  
  
    p = *pila;  
    *pila = (*pila)->succ;  
    free(p);  
  
    return 1;  
}
```

Implementazione in C (*cont.*)

// Esempi di invocazione

```
PPila p = NULL;  
int    n;
```

```
Push(&p, 1);  
Push(&p, 5);  
Push(&p, 4);
```

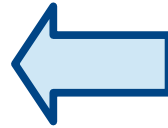
```
if (!Pop(&p, &n))  
    printf("La pila è vuota\n");
```

Coda

- Struttura FIFO
 - First In First Output
- Gli inserimenti vengono effettuati in coda

Esempio – Coda

```
Inserisci( 'C' );  
Inserisci( 'i' );  
Inserisci( 'a' );  
Inserisci( 'o' );  
c = Estrai();  
c = Estrai();  
c = Estrai();
```



c



Esempio – Coda

```
Inserisci( 'C' );
```

```
Inserisci( 'i' );
```

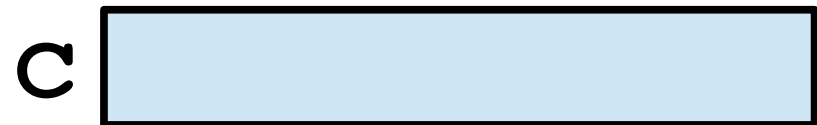
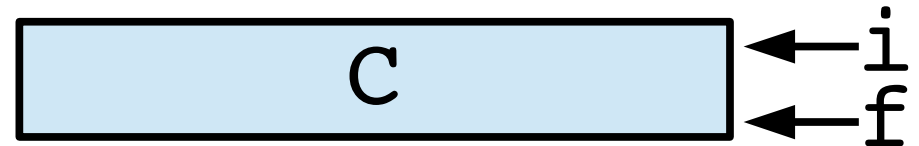
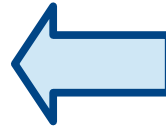
```
Inserisci( 'a' );
```

```
Inserisci( 'o' );
```

```
c = Estrai();
```

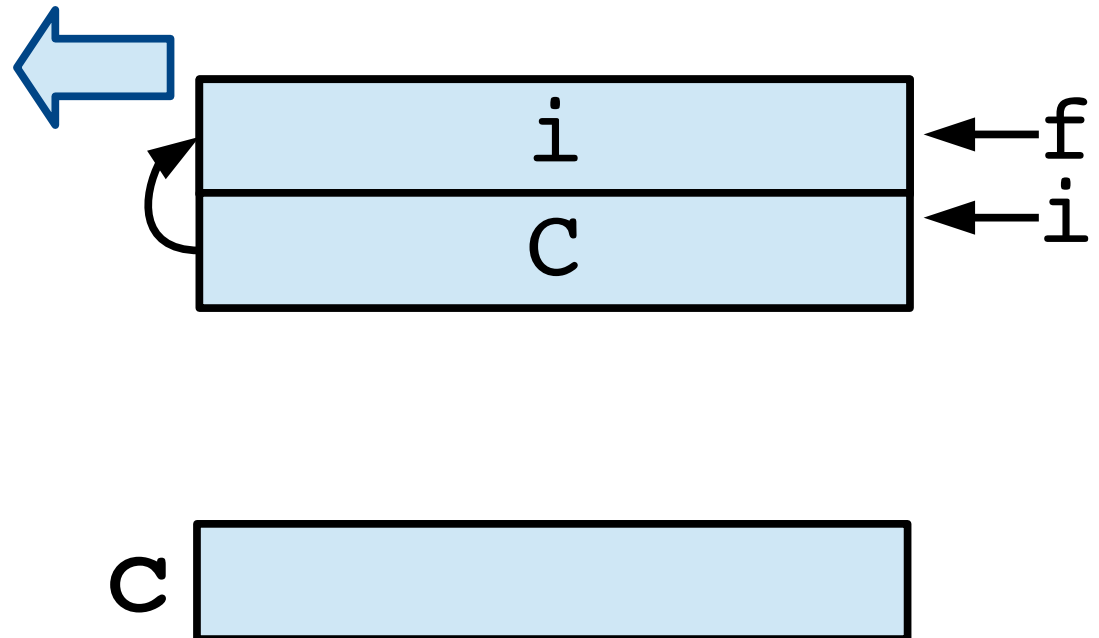
```
c = Estrai();
```

```
c = Estrai();
```



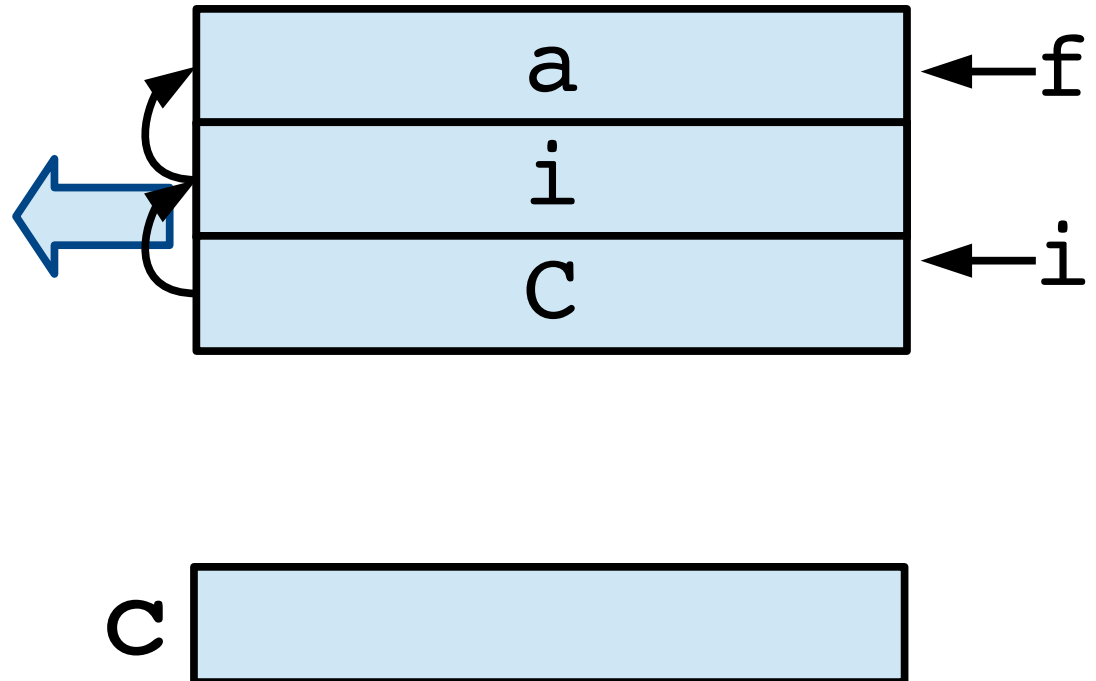
Esempio – Coda

```
Inserisci( 'C' );  
Inserisci( 'i' );  
Inserisci( 'a' );  
Inserisci( 'o' );  
c = Estrai();  
c = Estrai();  
c = Estrai();
```



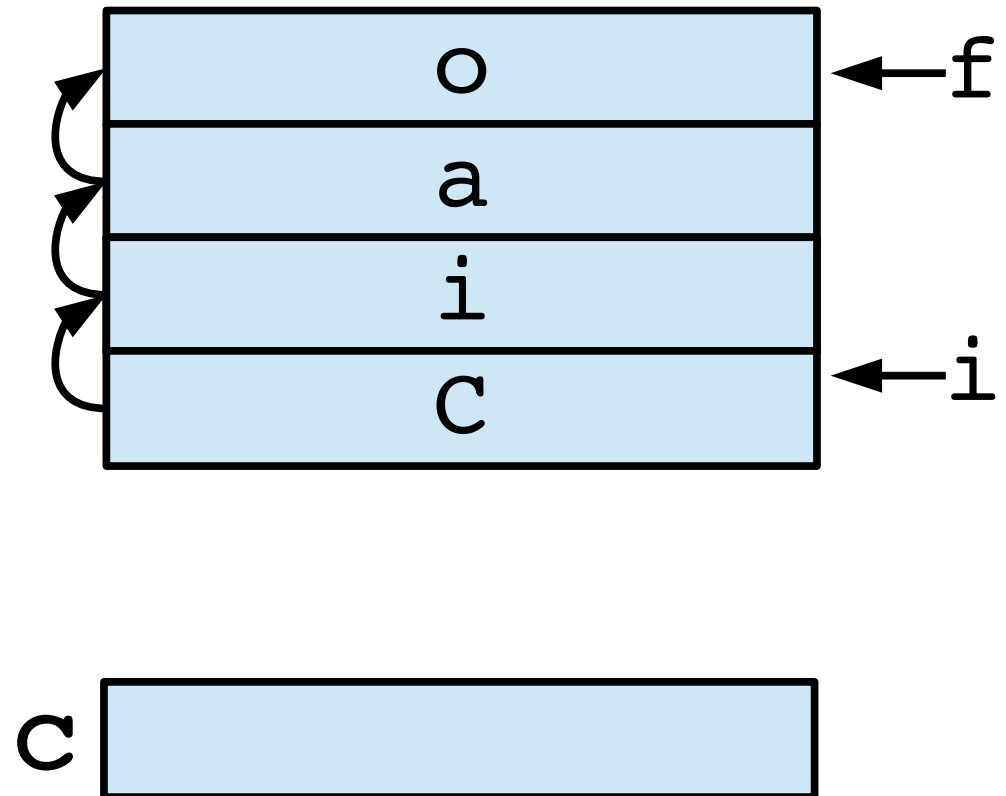
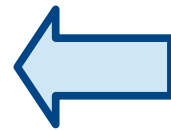
Esempio – Coda

```
Inserisci( 'C' );  
Inserisci( 'i' );  
Inserisci( 'a' );  
Inserisci( 'o' );  
c = Estrai();  
c = Estrai();  
c = Estrai();
```



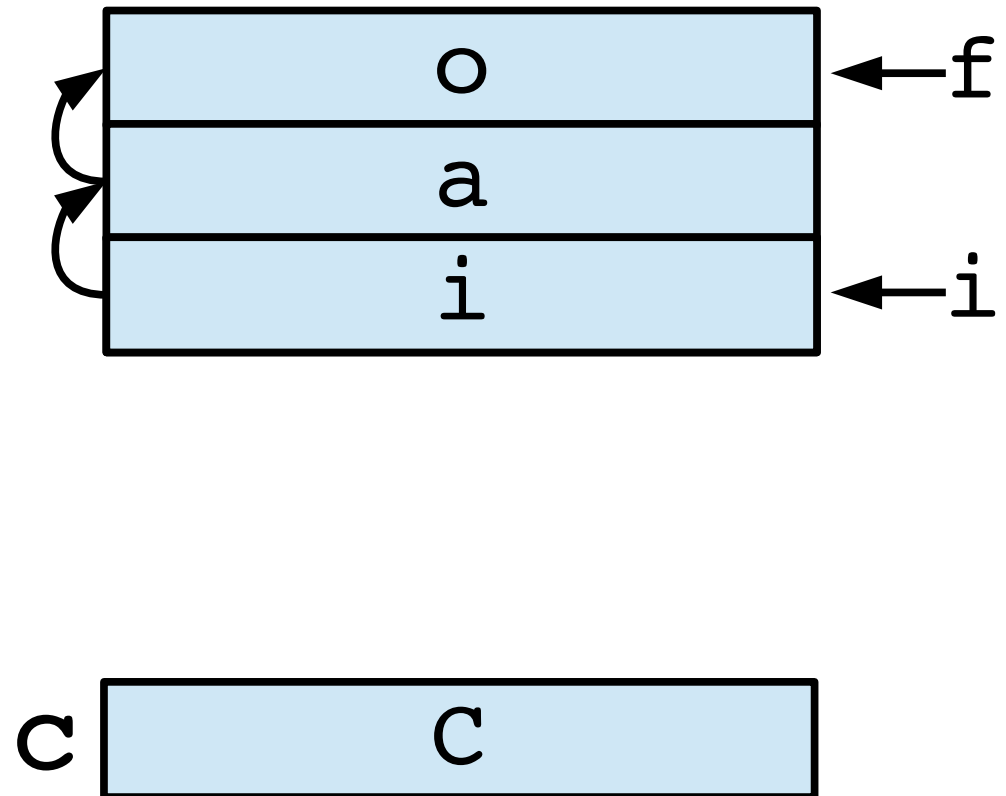
Esempio – Coda

```
Inserisci( 'C' );  
Inserisci( 'i' );  
Inserisci( 'a' );  
Inserisci( 'o' );  
c = Estrai();  
c = Estrai();  
c = Estrai();
```



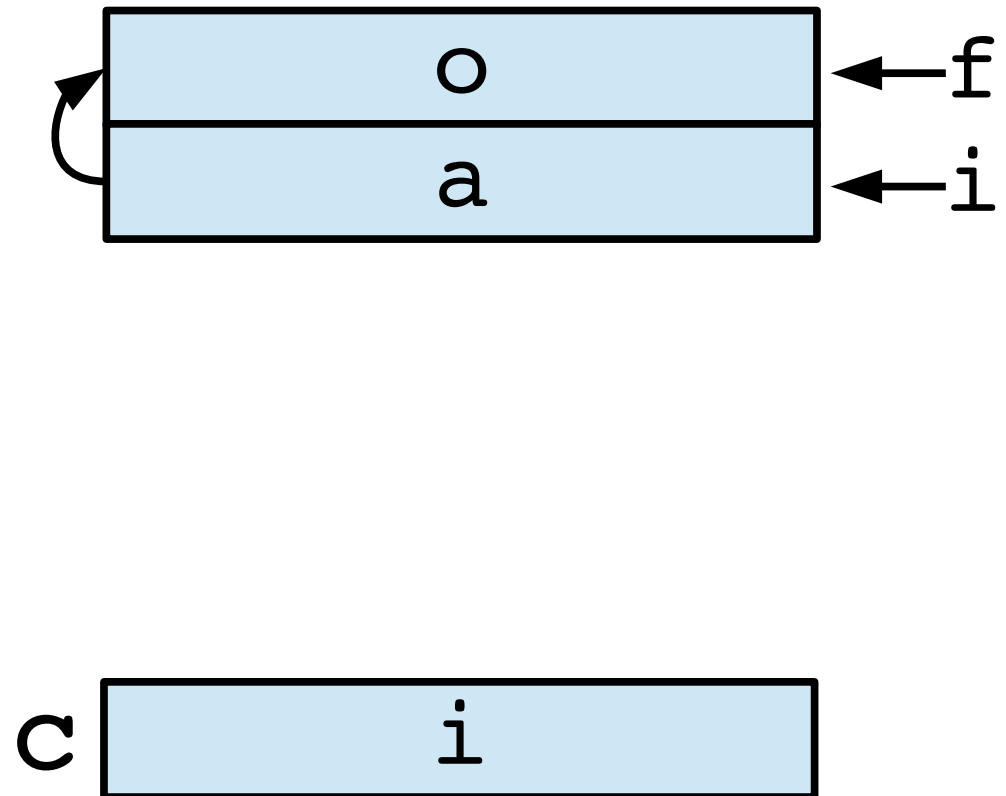
Esempio – Coda

```
Inserisci( 'C' );  
Inserisci( 'i' );  
Inserisci( 'a' );  
Inserisci( 'o' );  
c = Estrai();  
c = Estrai();  
c = Estrai();
```



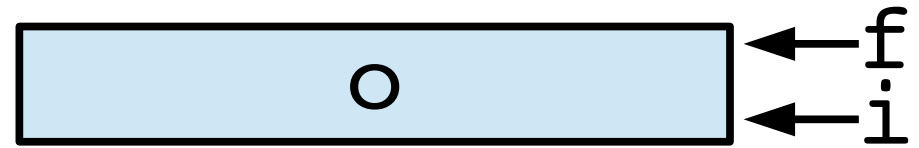
Esempio – Coda

```
Inserisci( 'C' );  
Inserisci( 'i' );  
Inserisci( 'a' );  
Inserisci( 'o' );  
c = Estrai();  
c = Estrai();  
c = Estrai();
```



Esempio – Coda

```
Inserisci( 'C' );  
Inserisci( 'i' );  
Inserisci( 'a' );  
Inserisci( 'o' );  
c = Estrai();  
c = Estrai();  
c = Estrai();
```



Implementazione in C

```
// Struttura dati
typedef struct Nodo
{
    int          info;
    struct Nodo* succ;
} TNode;

typedef TNode* PNode;

typedef struct Coda
{
    PNode primo;
    PNode ultimo;
} TCoda;
```

Implementazione in C (*cont.*)

```
// Inserisce n in coda
void Inserisci(TCoda* coda, int n)
{
    TNode* p;

    p = (TNode *)malloc(sizeof(TNode));
    p->info = n;
    p->succ = NULL;

    if (coda->ultimo != NULL)
        coda->ultimo->succ = p;
    else
        coda->primo = p;

    coda->ultimo = p;
}
```

Implementazione in C (*cont.*)

```
// Estrae il primo elemento della coda
int Estrai(TCoda* coda, int* n)
{
    TNode* p;

    if (coda->primo == NULL)
        return 0;

    *n = coda->primo->info;

    p = coda->primo;
    coda->primo = coda->primo->succ;
    free(p);

    return 1;
}
```

Implementazione in C (*cont.*)

```
// Esempi di invocazione
```

```
TCoda coda;  
int    n;
```

```
coda.primo  = NULL;  
coda.ultimo = NULL;
```

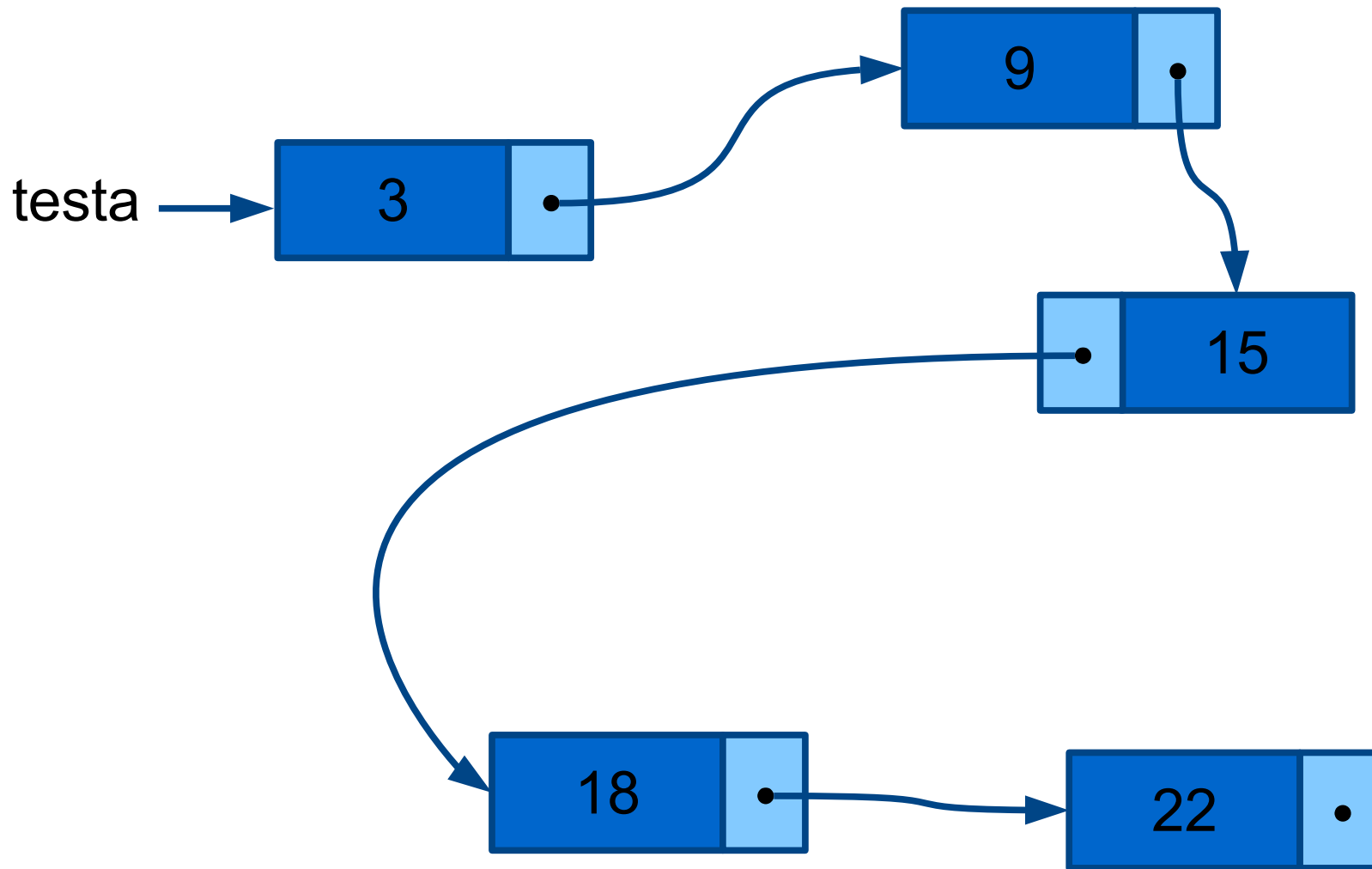
```
Inserisci(&coda, 3);  
Inserisci(&coda, 5);  
Inserisci(&coda, 7);
```

```
if (!Estrai(&coda, &n))  
    printf("La coda è vuota\n");
```

Inserimento Ordinato

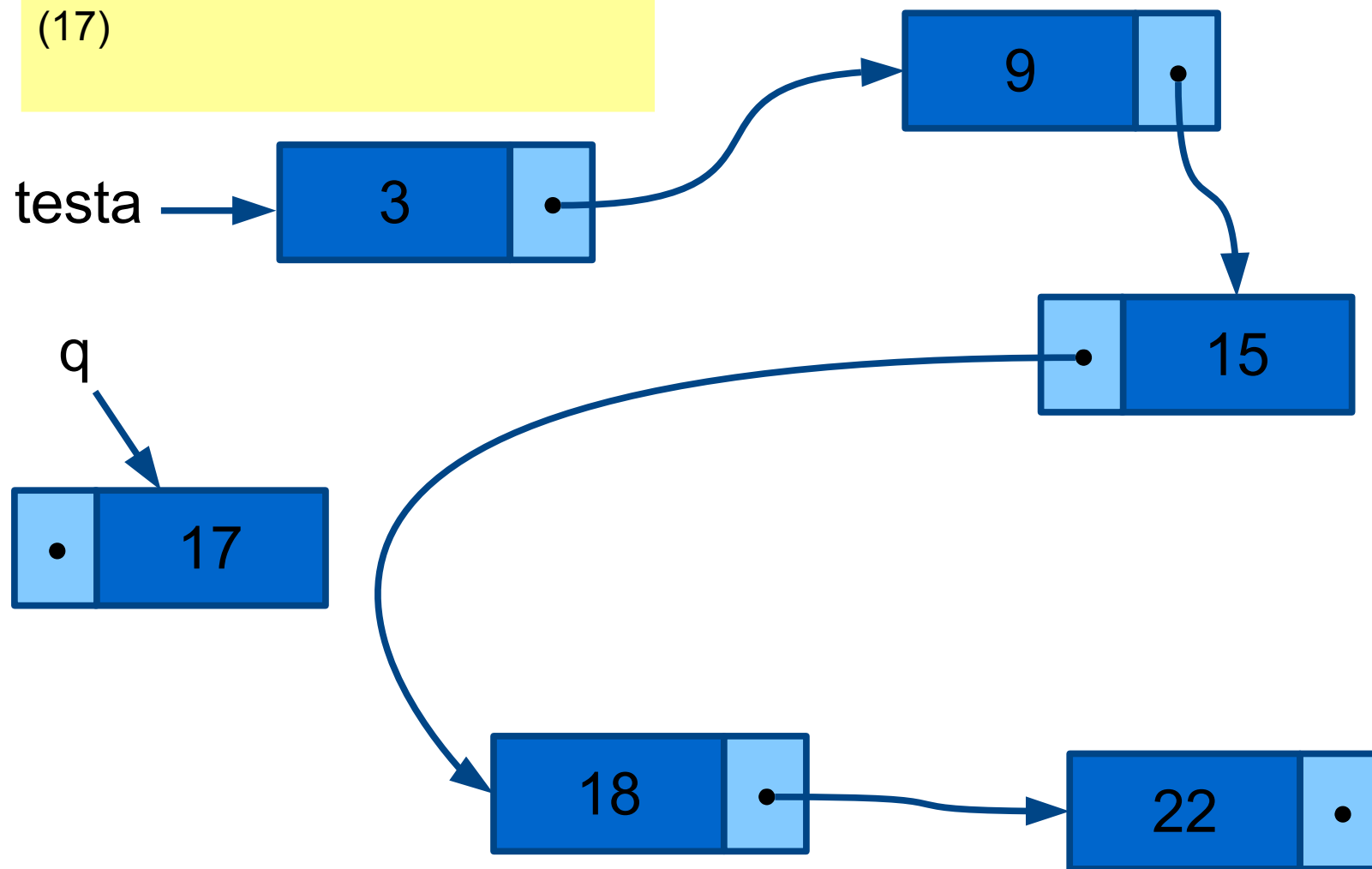
- Inserimento in lista mantenendo la lista ordinata

Inserimento Ordinato – Esempio



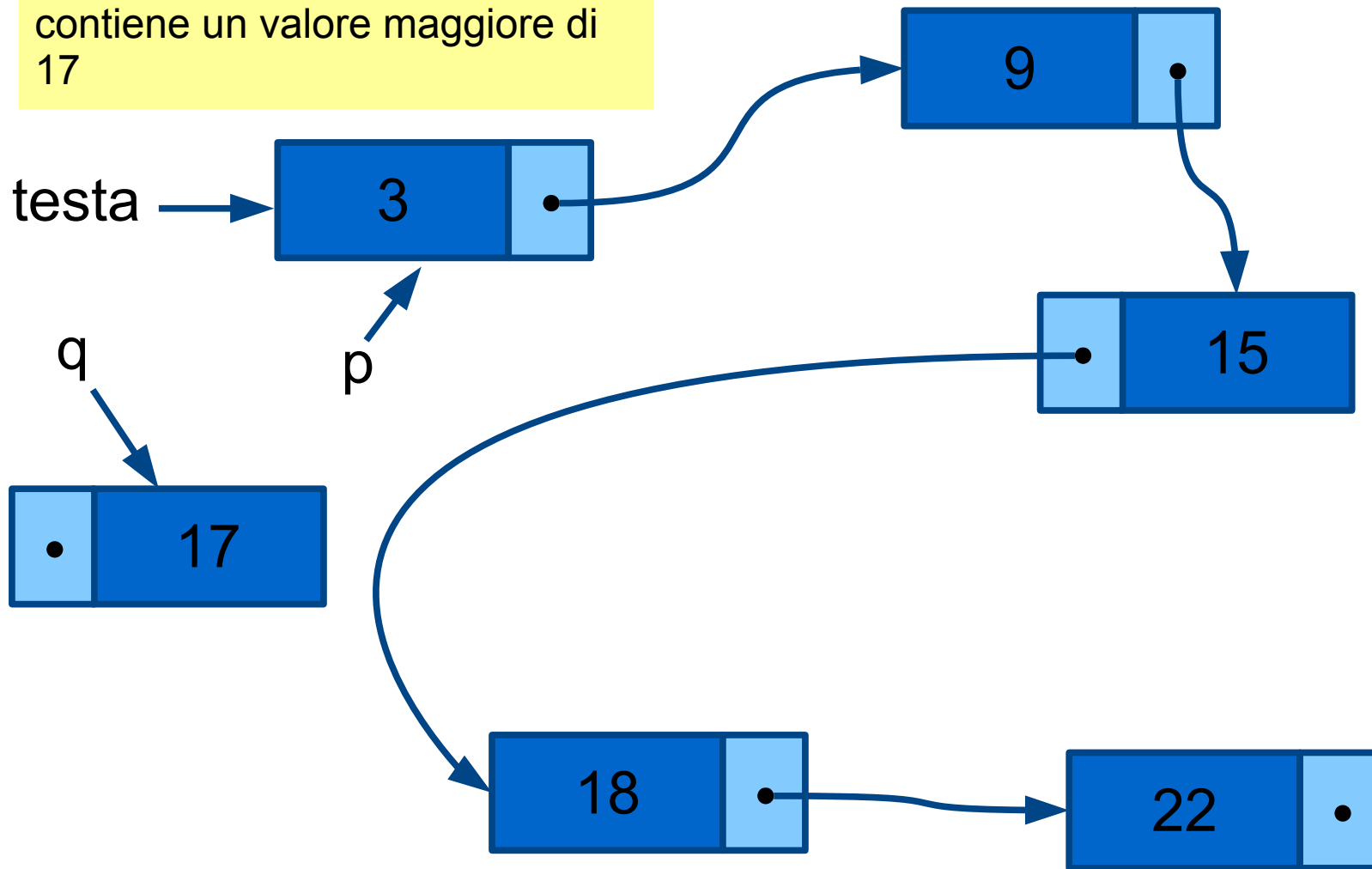
Inserimento Ordinato – Esempio

Creo il nuovo nodo da inserire (17)



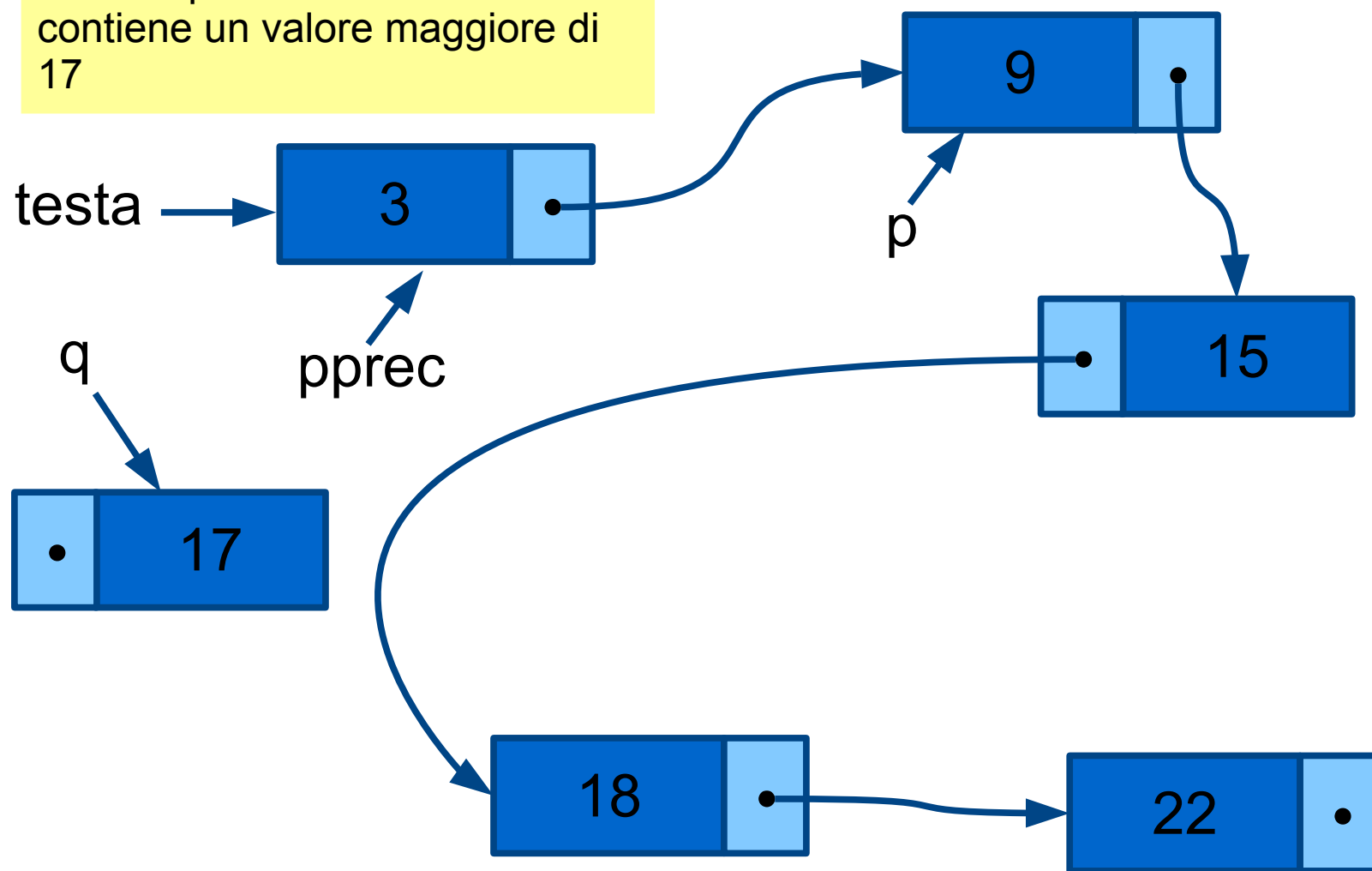
Inserimento Ordinato – Esempio

Cerco il primo nodo che
contiene un valore maggiore di
17



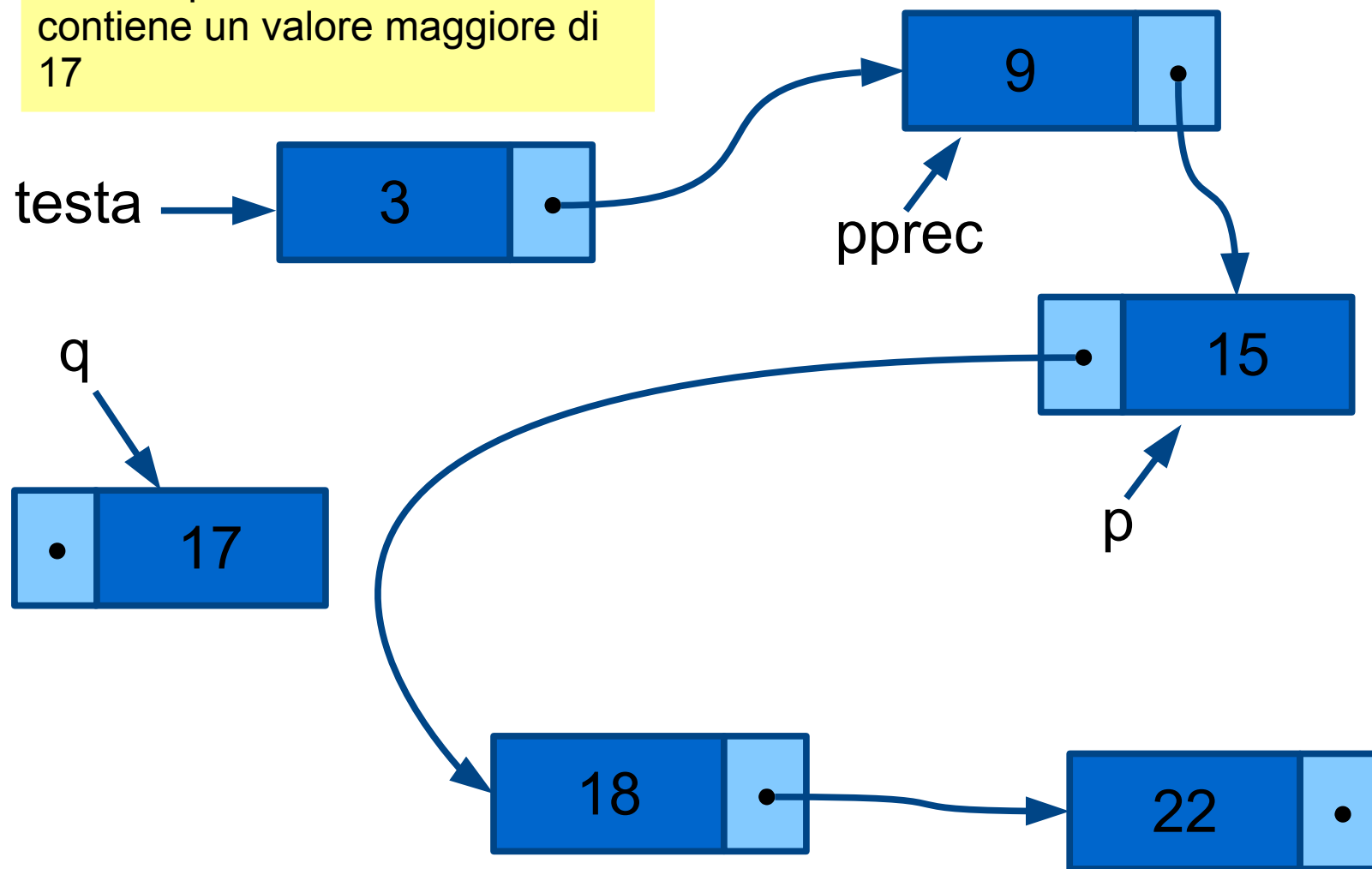
Inserimento Ordinato – Esempio

Cerco il primo nodo che contiene un valore maggiore di 17



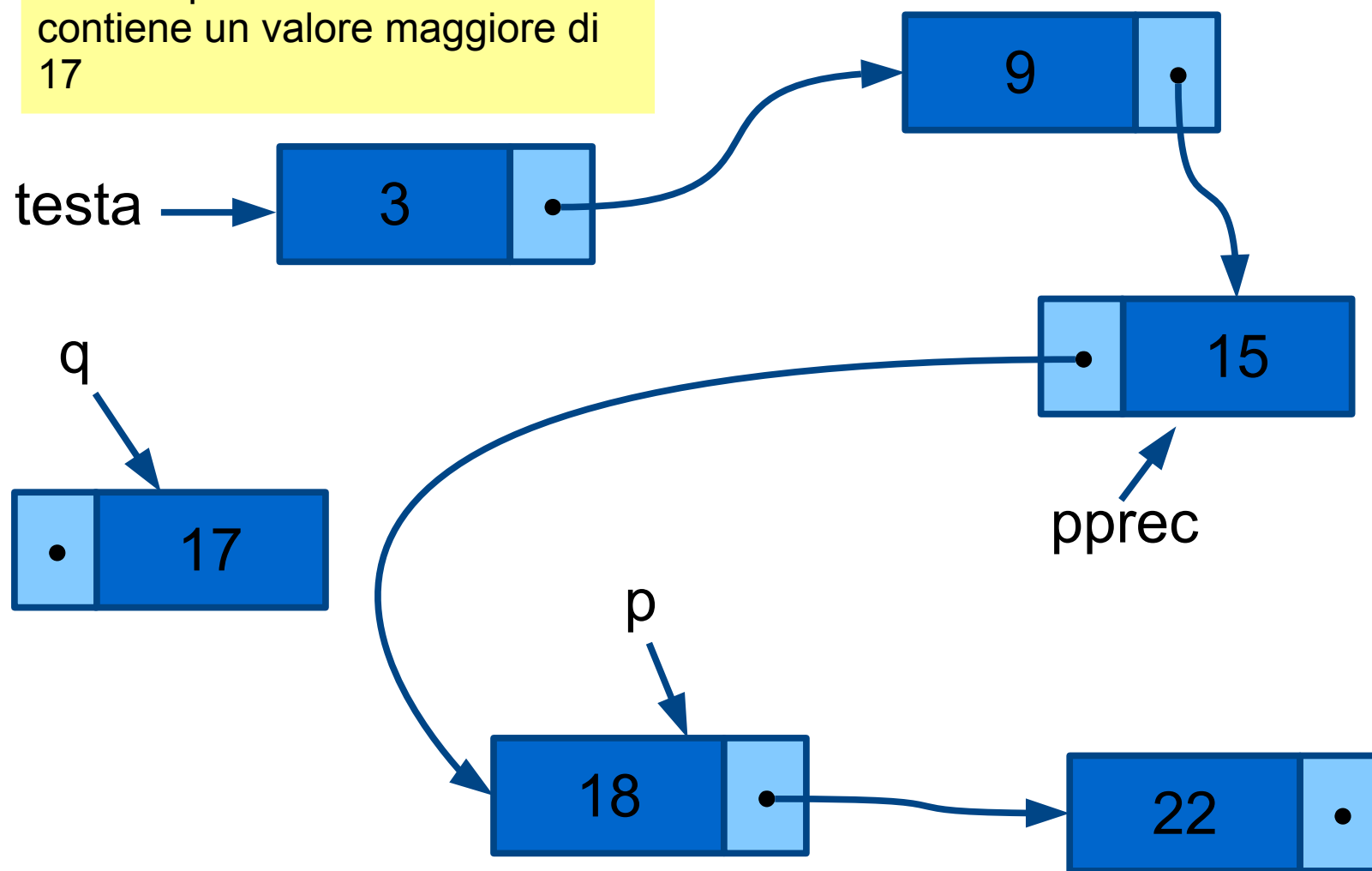
Inserimento Ordinato – Esempio

Cerco il primo nodo che contiene un valore maggiore di 17



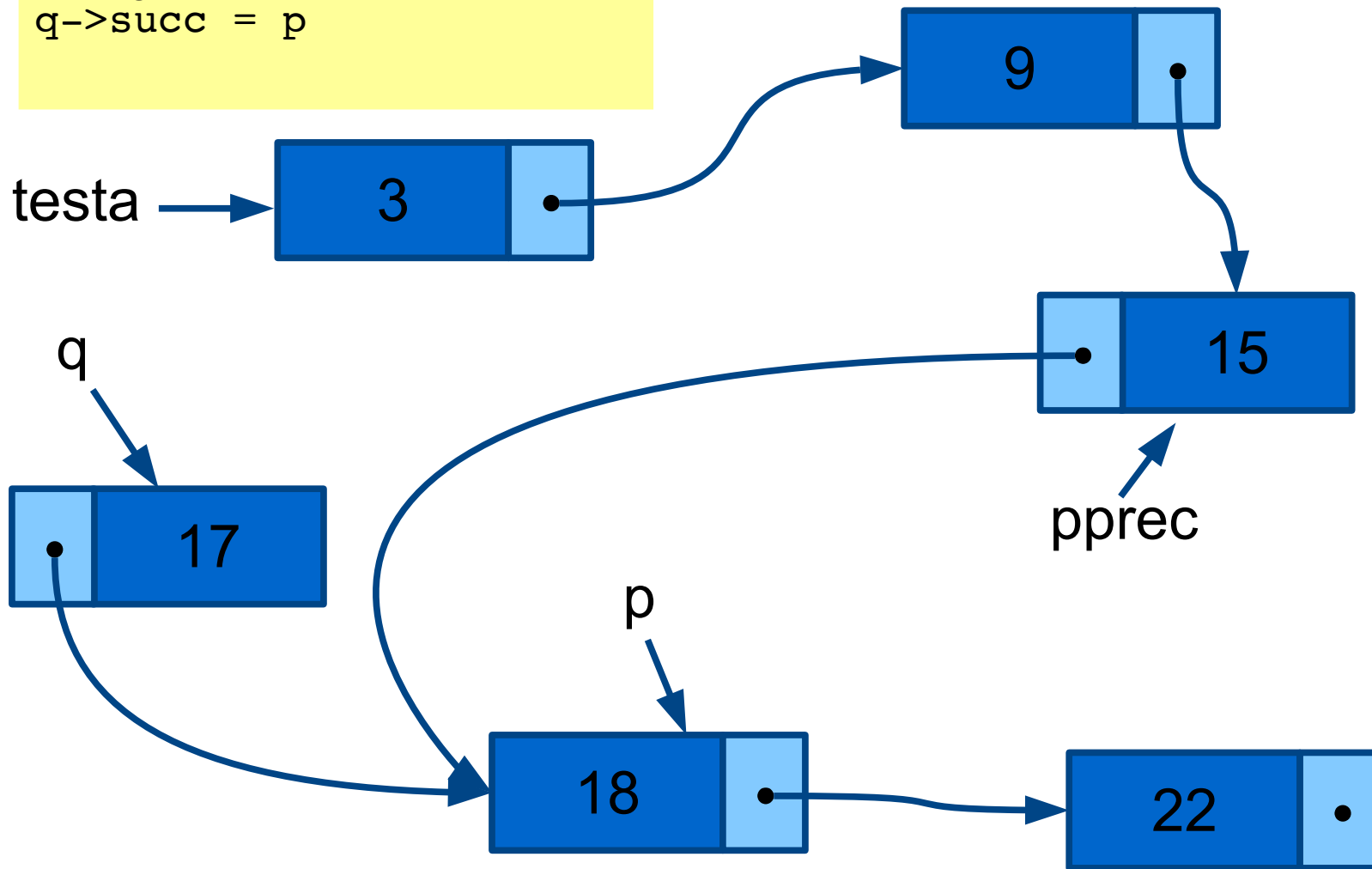
Inserimento Ordinato – Esempio

Cerco il primo nodo che contiene un valore maggiore di 17



Inserimento Ordinato – Esempio

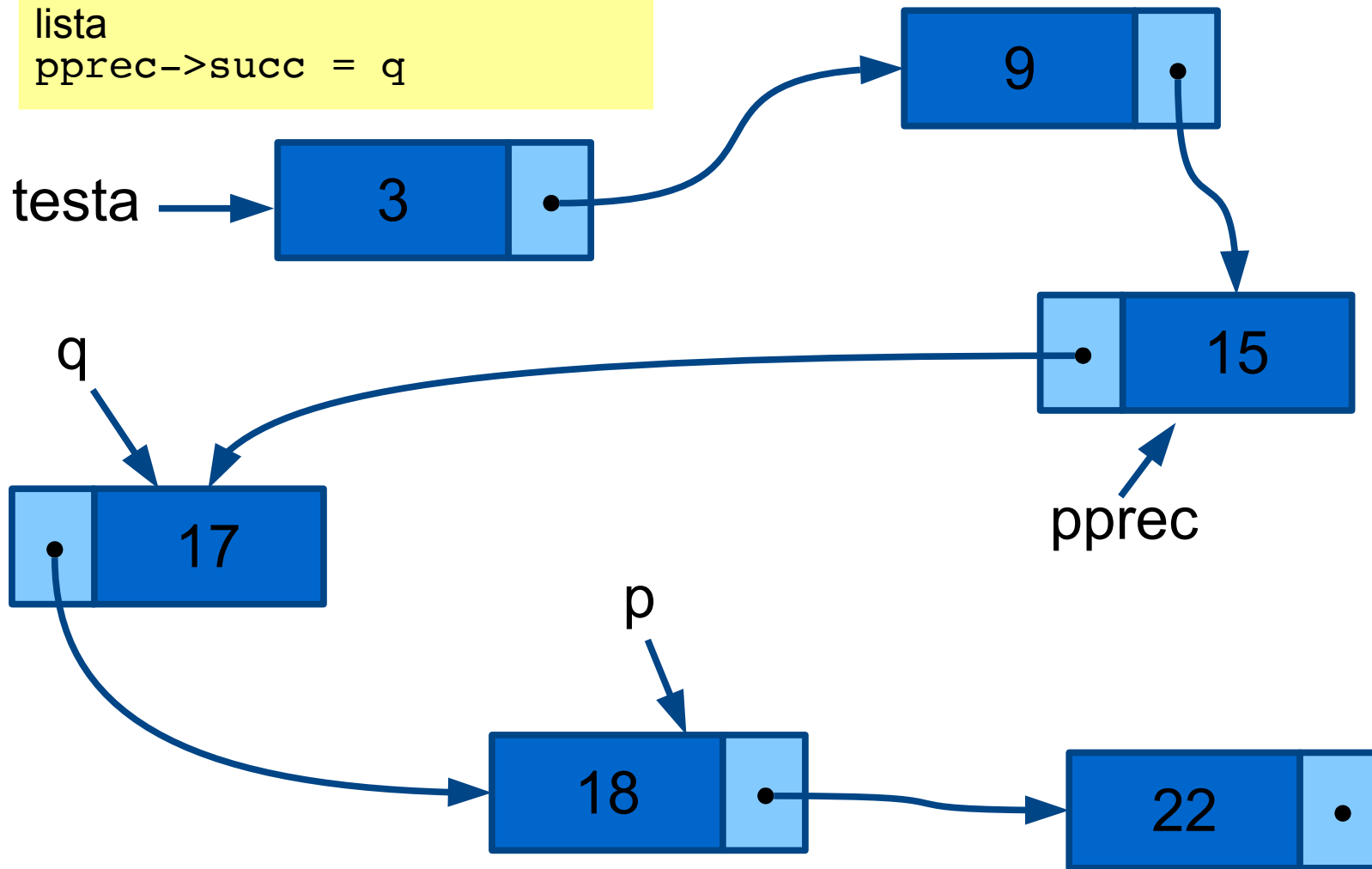
Collego il nuovo nodo alla lista:
 $q \rightarrow \text{succ} = p$



Inserimento Ordinato – Esempio

Inserisco il nuovo nodo nella lista

```
pprec->succ = q
```



Inserimento Ordinato (v1)

```
// Struttura dati
typedef struct Nodo
{
    int          info;
    struct Nodo* succ;
} TNode;

typedef TNode* PNode;
```

Inserimento Ordinato (v1)

```
PNode InserimentoOrdinato(PNode testa, int n)
{
    PNode q, p, pprec;

    // Creo il nuovo nodo
    q = (PNode)malloc(sizeof(TNode));
    q->info = n;

    // Cerco il primo nodo che contiene un valore
    // maggiore di n
    p = testa;
    pprec = NULL;
    while (p != NULL && p->info < n)
    {
        pprec = p;
        p = p->succ;
    }
}
```

Inserimento Ordinato (v1)

```
if (pprec == NULL)
    // Inserimento in testa
    testa = q;
else
    // Inserimento in mezzo o alla fine
    pprec->succ = q;

// Collego il nuovo nodo alla lista
q->succ = p;

// Restituisco la (nuova) testa della lista
return testa;
}
```

Inserimento Ordinato (v1)

// Esempi di invocazione

```
PNode l = NULL;
```

```
l = InserimentoOrdinato(l, 5);
```

```
l = InserimentoOrdinato(l, 1);
```

```
l = InserimentoOrdinato(l, 7);
```

```
l = InserimentoOrdinato(l, 50);
```

```
l = InserimentoOrdinato(l, 24);
```

Inserimento Ordinato (v2)

```
// Struttura dati
typedef struct Nodo
{
    int          info;
    struct Nodo* succ;
} TNode;

typedef TNode* PNode;
```

Inserimento Ordinato (v2)

```
void InserimentoOrdinato(PNodo* testa, int n)
{
    PNodo q, p, pprec;

    // Creo il nuovo nodo
    q = (PNodo)malloc(sizeof(TNodo));
    q->info = n;

    // Cerco il primo nodo che contiene un valore
    // maggiore di n
    p = *testa;
    pprec = NULL;
    while (p != NULL && p->info < n)
    {
        pprec = p;
        p = p->succ;
    }
}
```

Inserimento Ordinato (v2)

```
if (pprec == NULL)
    // Inserimento in testa
    *testa = q;
else
    // Inserimento in mezzo o alla fine
    pprec->succ = q;

// Collego il nuovo nodo alla lista
q->succ = p;
}
```

Inserimento Ordinato (v2)

// Esempi di invocazione

```
PNodo l = NULL;
```

```
InserimentoOrdinato(&l, 5);
```

```
InserimentoOrdinato(&l, 1);
```

```
InserimentoOrdinato(&l, 7);
```

```
InserimentoOrdinato(&l, 50);
```

```
InserimentoOrdinato(&l, 24);
```