



Liste



Fulvio CORNO - Matteo SONZA REORDA
Dip. Automatica e Informatica
Politecnico di Torino

Introduzione

Una **lista** è una struttura dati basata sull'uso dei puntatori e dell'allocazione/deallocazione dinamica della memoria.

Permette un maggiore flessibilità nell'uso della memoria rispetto ad altre strutture dati (ad es. i vettori) al costo di una minore efficienza.

Definizione

Una lista è una struttura dati in cui

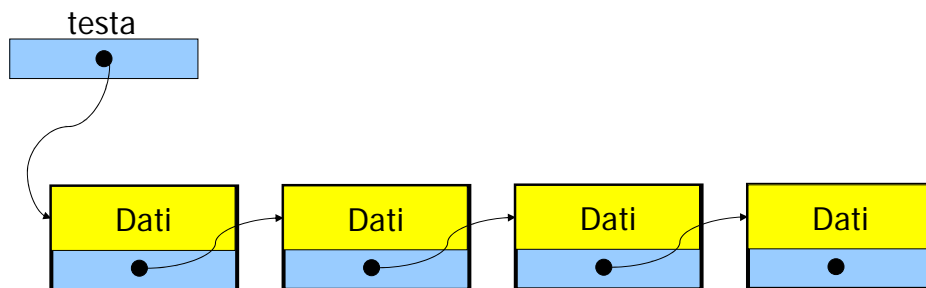
- Ogni elemento viene allocato/deallocato separatamente
- Ogni elemento è *linkato* agli altri (e quindi accessibile) attraverso puntatori
- Esiste una variabile (denominata *testa*) che permette di accedere al primo elemento.

A.A. 2001/2002

APA - Liste

3

Lista semplice



A.A. 2001/2002

APA - Liste

4



Conseguenze

- Ad ogni istante si utilizza solo la memoria corrispondente agli elementi effettivamente utilizzati
- Per accedere a ciascun elemento è necessario percorrere la lista.

A.A. 2001/2002

APA - Liste

5



Varianti

Oltre alle liste semplici, sono talvolta utilizzate

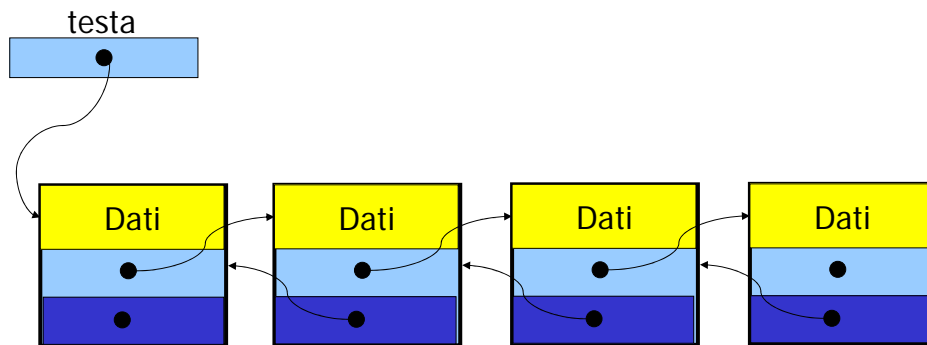
- Le liste **circolari**, in cui l'ultimo elemento contiene il puntatore al primo
- Le liste **con doppio puntatore** (uno all'elemento successivo, l'altro a quello precedente).

A.A. 2001/2002

APA - Liste

6

Liste con doppio puntatore



A.A. 2001/2002

APA - Liste

7

Implementazione in C

```
/* definizione */
struct list_el{
    int codice;
    char nome[20];
    char cognome[20];
    struct list_el *succ;
}

/* definizione testa */
struct list_el *testa=NULL;
```

A.A. 2001/2002

APA - Liste

8

Inserimento in testa

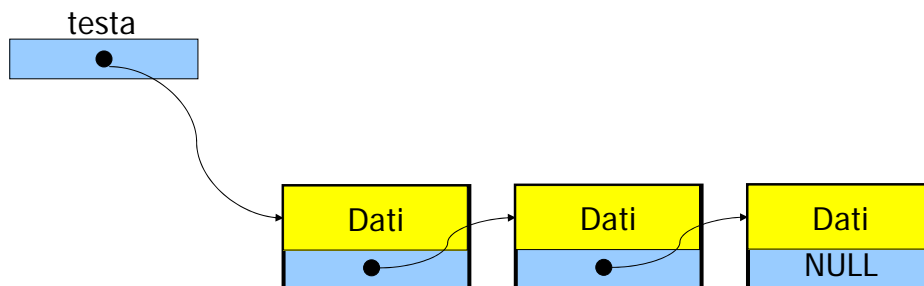
```
...  
struct list_el *p;  
...  
p=(struct list_el *)malloc(sizeof(struct list_el));  
if (p==NULL)  
    ERRORE  
p->codice=val;  
p->nome=strdup(nome);  
p->cognome=strdup(cognome);  
p->succ=testa;  
testa=p;  
...
```

A.A. 2001/2002

APA - Liste

9

Inserimento in testa (prima)

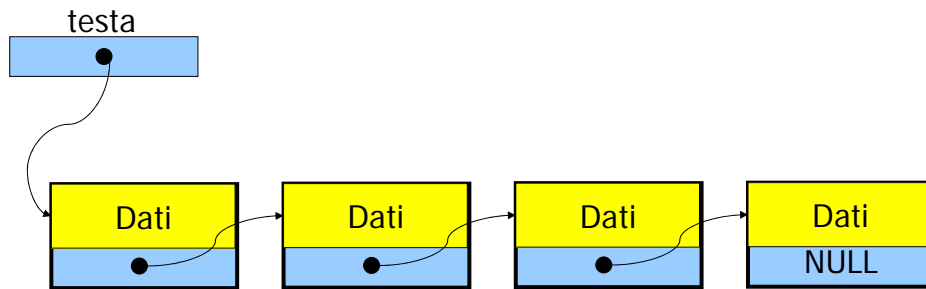


A.A. 2001/2002

APA - Liste

10

Inserimento in testa (dopo)



A.A. 2001/2002

APA - Liste

11

Procedura inserisci

```
int inserisci (struct list_el **t, int val, char *nome,
              char *cogn)
{
    struct list_el *p;
    p=(struct list_el *)malloc(sizeof(struct list_el));
    if (p==NULL)
        return (-1);
    p->codice=val;
    p->nome=strdup(nome);
    p->cognome=strdup(cognome);
    p->succ=*t;
    *t=p;
    return (0);
}
```

A.A. 2001/2002

APA - Liste

12



Programma chiamante

```
...
struct list_el *testa;
int ret, val;
char nome[MAX], cognome[MAX];
...
scanf ("%d, %s %s\n", &val, nome, cognome);
ret=inserisci (&testa, val, nome, cognome);
if (ret == -1)
    ERRORE
```

A.A. 2001/2002

APA - Liste

13



Ricerca

```
struct list_el *ricerca (struct list_el *t, int val)
{ struct list_el *p;
  p=t;
  while (p!=NULL)
  { if (p->codice==val)
    return (p);
    p=p->succ;
  }
  return (p);
}
```

A.A. 2001/2002

APA - Liste

14



Programma chiamante

```
...
struct list_el *testa, *p;
int val;
...
scanf ("%d\n", &val);
p=ricerca (testa, val);
if (p == NULL)
    printf ("Elemento non trovato\n");
else
    printf ("%d %s %s\n", p->codice, p->nome, p->cogn);
...
```

A.A. 2001/2002

APA - Liste

15



Liste ordinate

Se la procedura di inserimento inserisce il nuovo elemento nella posizione opportuna, la lista è mantenuta ordinata (rispetto ad un certo campo chiave).

In tal modo è possibile

- Semplificare le operazioni di ricerca
- Accedere agli elementi in ordine.

A.A. 2001/2002

APA - Liste

16

Inserimento in lista ordinata

```
int inserisci_ord (struct list_el **t, int val, char *nome,
                  char *cogn)
{
    struct list_el *p, *q;
    /* allocazione nuovo elemento */
    p=(struct list_el *)malloc(sizeof(struct list_el));
    if (p==NULL)
        return (-1);
    p->codice=val;
    p->nome=strdup(nome);
    p->cogn=strdup(cogn);
    q = *t;
    /* inserimento in testa */
    if( (q == NULL) || (q->codice > val))
    {
        p->succ=*t;
        *t=p;
        return (0);
    }
}
```

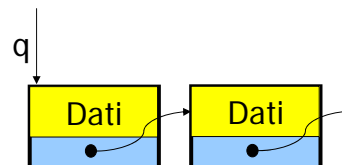
A.A. 2001/2002

APA - Liste

17

Inserimento in lista ordinata (2)

```
/* inserimento in mezzo */
while( q->succ != NULL)
{
    if( q->succ->codice > val)
    {
        p->succ=q->succ;
        q->succ=p;
        return (0);
    }
    q = q->succ;
}
/* inserimento in coda */
p->succ = NULL;
q->succ = p;
return( 0);
}
```



A.A. 2001/2002

APA - Liste

18



Cancellazione

La cancellazione di un elemento normalmente richiede

- Un'operazione di **ricerca**, che produce il puntatore all'elemento da cancellare
- Un'operazione di **cancellazione** vera e propria. Questa richiede non solo il puntatore all'elemento da cancellare, ma anche quello all'elemento precedente.

A.A. 2001/2002

APA - Liste

19



Cancellazione

```
int cancella (struct list_el **t, int val)
{ struct list_el *p, *q;
  q = *t;
  if (q==NULL)      /* lista vuota */
    return (-1);
  /* cancellazione in testa */
  if (q->codice > val)
  { free (q->nome);
    free (q->cogn);
    *t=q->succ;
    free (q);
    return (0);
  }
}
```

A.A. 2001/2002

APA - Liste

20



Cancellazione (2)

```
/* cancellazione in mezzo o in coda */
while( q->succ != NULL)
{  if( q->succ->codice == val)
    {  q->succ=q->succ->succ;
        free (q->nome);
        free (q->cogn);
        free (q);
        return (0);
    }
    q = q->succ;
}
```

A.A. 2001/2002

APA - Liste

21



Complessità

L'inserimento in testa ha una complessità pari a $O(1)$.

Tutte le altre operazioni sulle liste hanno una complessità $O(n)$, in quanto nel caso peggiore richiedono la visita di tutti gli elementi nella lista.

A.A. 2001/2002

APA - Liste

22



Sentinelle

Può essere conveniente aggiungere alla lista degli elementi fittizi (detti *sentinelle*) che permettono di

- Semplificare il codice per la gestione delle liste
- Aumentarne l'efficienza (senza cambiare la complessità nel caso peggiore).

A.A. 2001/2002

APA - Liste

23



Sentinelle: liste non ordinate

In questo caso si usano solitamente due sentinelle (una in coda ed una in testa).

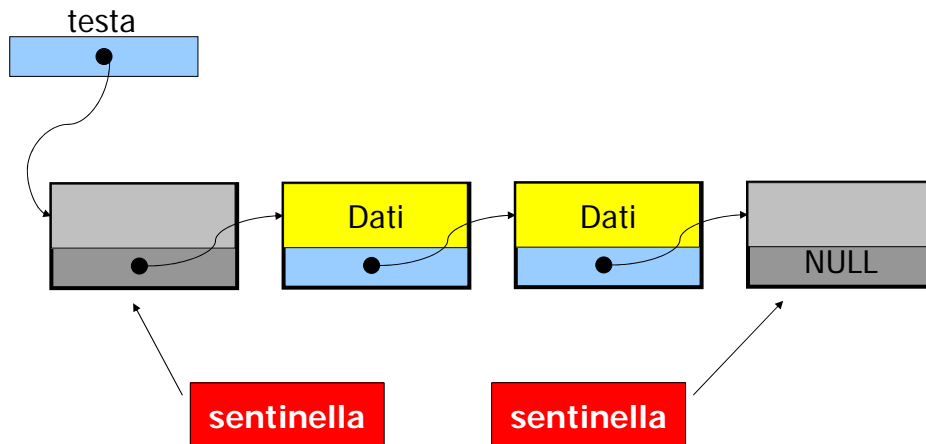
Le sentinelle permettono di **semplificare il codice** (non si devono più considerare separatamente i casi di inserimento/cancellazione in testa e in coda).

A.A. 2001/2002

APA - Liste

24

Esempio



A.A. 2001/2002

APA - Liste

25

Sentinelle: liste ordinate

In questo caso le due sentinelle contengono il valore minimo e quello massimo memorizzabili.

In tal modo

- Si semplifica il codice (non si devono più considerare separatamente i casi di inserimento/cancellazione in testa e in coda)
- Si rende più veloce l'esecuzione, in quanto si può eliminare il test di fine lista.


A.A. 2001/2002

APA - Liste

26

Maggiore velocità

```
struct list_el *ricerca (struct list_el *t, int val)
{ struct list_el *p;
  p=t;
  while (p!=NULL)
  { if (p->codice==val)
    return (p);
    p=p->succ;
  }
  return (p);
}
```



```
{ struct list_el *p;
  p=t;
  while (p->codice<=val)
    p=p->succ;
  if (p->codice==val)
    return (p);
  else
    return (NULL);
}
```

Ad ogni iterazione si eseguono due test

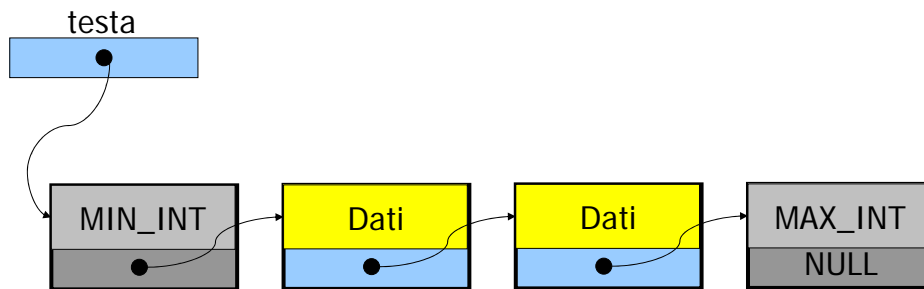
Ad ogni iterazione si esegue un solo test

A.A. 2001/2002

APA - Liste

27

Esempio



A.A. 2001/2002

APA - Liste

28