Liste concatenate

Una lista concatenata è una struttura dati in cui gli elementi (i dati) sono sistemati secondo un ordine lineare, così come negli array, ma a differenza degli array in cui l'ordine è mantenuto dagli indici dell'array stesso, in questo caso, invece, in ciascun elemento è mantenuta l'informazione su chi sia il prossimo elemento.

Tale tipo di organizzazione permette di realizzare operazioni di inserimento e cancellazione di elementi in maniera più facile (meno dispendiosa dal punto di vista dei tempi computazionali) rispetto ad un organizzazione con array. Ovviamente supponendo di voler conservare l'ordine degli elementi.

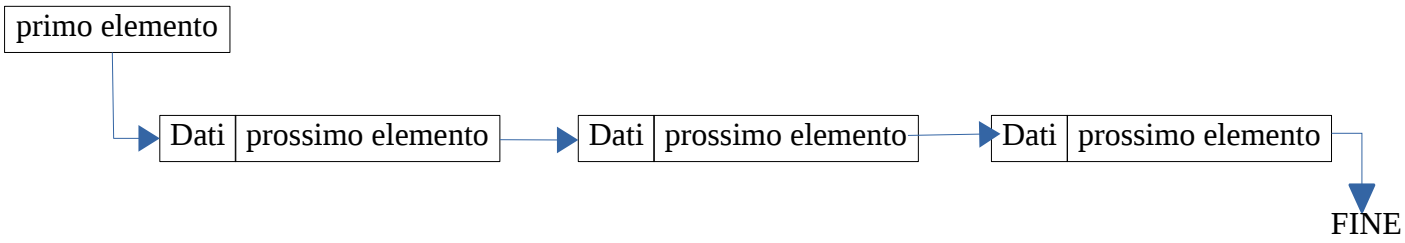
In un array, infatti, l'eliminazione di un elemento o l'inserimento di un elemento comporta lo spostamento di “molti” altri elementi se vogliamo conservare l'ordine. Ad esempio in un array che contiene 100 elementi se vogliamo inserire un elemento in 50-sima posizione dobbiamo spostare in avanti (“shiftare”) gli elementi che vanno dalla posizione 50 alla posizione 100. Con le liste concatenate operazioni di questo tipo vengono effettuate in maniera “locale”, cioè operando solo sugli elementi vicini alla posizione in cui volgiamo inserire un nuovo elemento o da cui vogliamo eliminare un elemento.

Possiamo subito dire che:

	VANTAGGI	SVANTAGGI
Array	Accesso diretto ai dati (tramite l'indice dell'elemento)	Operazioni di inserimento e cancellazione quando l'ordine degli elementi è preservato
Liste Concatenate	Operazioni di inserimento e cancellazione quando l'ordine degli elementi è preservato	Accesso sequenziale ai dati

La possibilità con le liste concatenate di effettuare in maniera efficace operazioni di cancellazione ed inserimento rende tale struttura dati adatta ad immagazzinare informazioni che cambiamo sostanzialmente nel tempo. Per tale motivo tale struttura dati fa parte di quelle strutture dati che vengono dette **strutture dati dinamiche**.

Come già affermato in una lista concatenata ciascun elemento mantiene sia l'informazione riguardante i dati sia l'informazione su chi è il prossimo elemento. Nella seguente figura e' riportato un esempio figurativo di lista concatenata.



Come si vede dalla figura, in una lista concatenata è necessario anche mantenere l'informazione su chi è il primo elemento, e l'ultimo elemento deve essere “riconoscibile”. Cioè l'ultimo elemento della lista deve avere come 'prossimo elemento' un valore che indichi che non c'è più nulla dopo.

Possiamo riassumere, allora, che una lista concatenata è caratterizzata da:

- gli elementi (i dati) sono sistemati secondo un ordine lineare,
- i dati sono tutti dello stesso tipo,
- ciascun elemento deve contenere almeno due informazioni: una o più sui dati, una su chi è il prossimo elemento,
- si deve sempre mantenere l'informazione su chi è il primo elemento,
- l'ultimo elemento deve essere riconoscibile.

Si noti che affinché sia possibile mantenere in maniera non ambigua l'informazione su chi è il primo elemento oppure su chi è il prossimo elemento, ciascun elemento della lista deve essere identificato in maniera univoca.

Una naturale implementazione in C delle liste concatenate si può realizzare usando le struct ed i puntatori fisici.

Vediamo come.

Implementazione in C delle liste concatenate

Ciascun elemento di una lista può essere visto come una struct contenete uno o più campi che mantengono l'informazione sui dati, ed un campo mantenente l'informazione su chi è il prossimo elemento, cioè l'indirizzo di memoria della struct rappresentate il prossimo elemento. Si osservi che in questo modo ciascun elemento della lista è identificato dall'indirizzo di memoria della struct che è **univoco**. Ad esempio:

/* Al di fuori del main:*/

```
struct elem {
      int dati; /*Campo o campi in cui sono mantenuti i dati
                      (l'informazione ) */
      struct elem *next; /*Puntatore al prossimo elemento */
};
```

/* Nel corpo del main o nel corpo di una funzione:*/

```
struct elem *top=NULL; /* In questo modo top sara' un puntatore ad una
                          lista inizialmente vuota*/
```

In questo caso 'struct elem' rappresenta il “tipo” di ciascun elemento della lista. La variabile 'top' rappresenta una variabile che conterrà sempre l'indirizzo di memoria del primo elemento della lista. Tale variabile inizialmente è pari a NULL in quanto la lista è vuota (non c'è nessun elemento). L'ultimo elemento della lista, se esiste, sarà riconoscibile in quanto il suo campo next sarà pari a NULL. Si osservi che è **fondamentale** fare in modo che ci sia sempre una variabile che contenga l'indirizzo di memoria del primo elemento della lista (altrimenti la lista è “persa”, non riusciremo più ad accedere ad essa), e che l'ultimo elemento, se esite, abbia il campo 'next' pari a NULL.

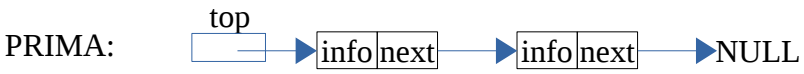
Vediamo ora come realizzare alcune delle principali operazioni sulle liste concatenate.

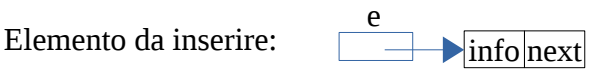
Creazione di un elemento

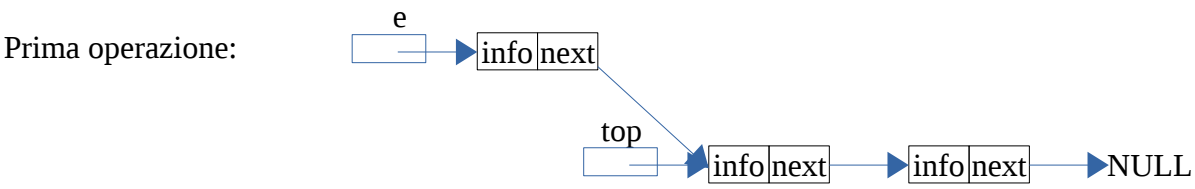
```
struct elem *newElem(int x){
      struct elem *tmp=(struct lista *)malloc(sizeof(struct elem));
      tmp->dati=x;
      tmp->next=NULL;
      return tmp;
}
```

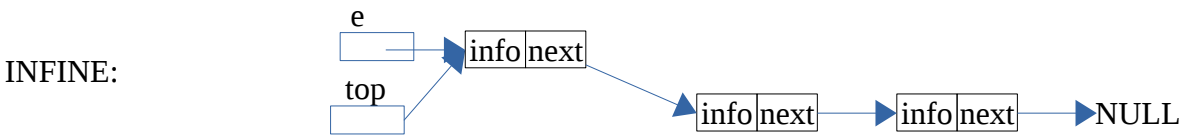
Inserimento di un elemento (`inserimento in testa`)

Graficamente vediamo un esempio di cosa accade:









Come codice, allora, possiamo scrivere quanto segue:

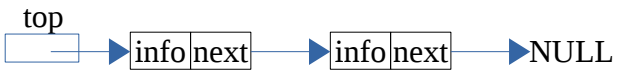
```
struct elem *insericiInTesta(struct elem *top, int x) {
      struct elem *e=newElem(x);
      e->next=top;
      return e;
}
```

In questo modo la funzione restituisce il nuovo top della lista ed un nuovo elemento è inserito come primo elemento della lista.

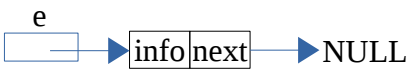
Inserimento di un elemento (`inserimento in coda`)

`Graficamente vediamo un esempio di cosa accade:`

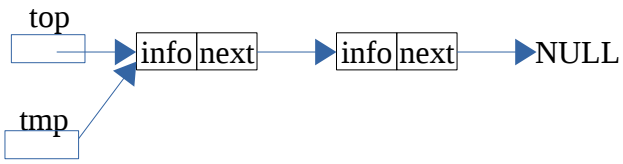
PRIMA:



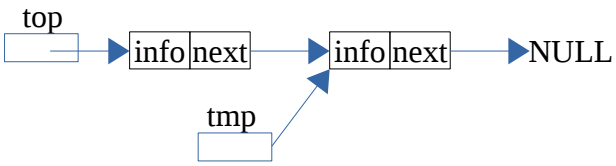
Elemento da inserire:



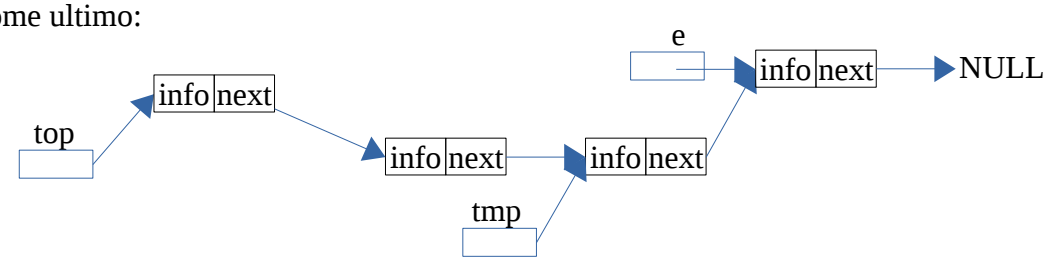
Prima operazione:



Si itera fino a far raggiungere a tmp l'ultimo elemento:



INFINE, si pone il nuovo elemento come ultimo:



Una possibile funzione C per fare tale inserimento è la seguente (versione iterativa):

```
struct elem *inserisciInCoda(struct elem *top, int x) {
      struct elem *tmp, *e;
      e=newElem(x);
      if (top!=NULL) {
            for (tmp=top;tmp->next != NULL;tmp=temp->next);
            /* Il for termina quando tmp->next e' NULL.
                  L'istruzione tmp=temp->next permette
                  di posizionarsi sul prossimo elemento
                  della lista.*/
            tmp->next=e;
      }
```

```
      else top=e;
      return top;
}
```

In questo caso un nuovo elemento è inserito come ultimo elemento della lista ed è restituito il nuovo top della lista. Si noti che il top cambia quando la lista iniziale è vuota

Cancellazione dell'intera lista concatenata (versione iterativa)

```
void cancellaLista(struct elem *top) {
      struct elem *tmp;
      for (;top!=NULL; top=tmp) { /* Itero su tutti gli elementi e li
                                     elimino;*/
            tmp=top->next;
            top->next=NULL;
            free(top);      /*Si noti che prima di eliminare un elemento
                              devo conservare il prossimo in tmp*/
      }
}
```

Di seguito una nota sull'uso della precedente funzione.

Se abbiamo una variabile top che mantiene una lista concatenata, per cancellare l'intera lista possiamo fare quanto segue:

```
cancellaLista(top);
top=NULL;
```

In altre parole bisogna assicurarsi che dopo aver eliminato tutti gli elementi della lista top sia messo a NULL.

Questo si può anche realizzare costruendo una funzione cancellaLista (o canellaListaR) che restituisca direttamente NULL, e poi chiamare:

```
top=cancellaLista(top);
```

Ricerca lineare di un elemento (versione iterativa)

In questo caso si suppone che gli elementi non siano necessariamente ordinati e che, quindi, è necessario una ricerca lineare (scandire tutti gli elementi dal primo all'ultimo).

```
struct elem *cercaElem(struct elem *top, int k) {
```

```
	struct elem *res=NULL;
	/*Itero su tutti gli elementi*/
	for (;top!=NULL && res==NULL; top=top->next)
		if (top->info==k) res=top;
	/* Si noti che top è una variabile locale, quindi anche se viene
cambiata in valore, non si modifica il valore della variabile che
mantiene il primo elemento della lista, che è stata passata come
parametro attuale*/
	return res;
}
```

Stampa di una lista (versione iterativa)

```
void stampaLista(struct elem *top) {
	while (top!=NULL) {
		printf("valore:%d\n", top->info);
		top=top->next;
	}
}
```

Utilizzando alcune delle funzioni precedenti vediamo un esempio di creazione, stampa e cancellazione di una lista:

```
struct elem {
	int info;
	struct elem *next;
};
int main() {
	int i,n,x;
	struct elem *top=NULL; /* top indica il primo elemento della
lista*/
	/* Costruzione di un lista di n elementi disposti secondo l'ordine
	con cui sono inseriti*/
	printf("Quanti elementi vuoi inserire?");
	scanf("%d",&n);
	for (i=0;i<n;++i){
		printf("Dammi l'elemento %d da inserire:",i);
		scanf("%d",&x);
```

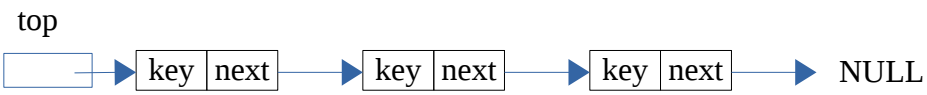
```
            top= inserisciInCoda(top,x);
        }
        stampaLista(top);
        cancellaLista(top);

return 0;
}
```

Inserimento “in mezzo” di un elemento.

Supponiamo, ora, di avere una lista concatenata con gli elementi ordinati secondo un campo chiave (cioè ciascun elemento può contenere anche più campi che mantengono diverse informazioni, tuttavia gli elementi della lista sono ordinati rispetto ad uno specifico campo, che è detto campo chiave). Chiamiamo tale campo 'key'.

Abbiamo, cioè, una situazione del genere:



Dove in ciascun elemento si è evidenziato solo il campo chiave, ma possono essere presenti anche altri campi.

Supponiamo, per semplicità, che il campo chiave sia un intero e che non ci siano altri campi. Supponiamo, inoltre, che gli elementi della lista siano ordinati in ordine crescente (il primo ha un campo chiave minore o uguale al secondo, il secondo ha un campo chiave minore o uguale al terzo, e così via).

Vogliamo inserire un nuovo intero elemento con campo chiave uguale a k nella lista concatenata facendo si che venga mantenuto l'ordine crescente. In questo caso il nuovo elemento k deve essere posto subito prima del primo elemento con valore del campo chiave maggiore di k.

Possono accadere le seguenti quattro situazioni alternative:

1) La lista è vuota. In questo caso è sufficiente creare un nuovo elemento e metterlo come unico elemento della lista.
2) Non ci sono elementi più grandi di k, allora il nuovo elemento è posto in coda alla lista.
3) Tutti gli elementi sono più grandi di k, allora k deve essere posto come top della lista.
4) k deve essere posto “in mezzo” tra due elementi, prima di un elemento più grande e dopo un elemento minore o uguale a k. In questa situazione ho bisogno di due puntatori che mi indichino i due elementi tra cui inserire k.

Si osservi che solo nei casi 1) e 3) cambia il top della lista. Il caso 2 è, in effetti, un situazione particolare

del caso 4.

Un possibile codice C per risolvere tale problema è il seguente (inserimento in ordine di un elemento con la lista già ordinata):

```
struct elem *inserisciInOrdine(struct elem *top, int k) {
      struct elem *e=newElem(k);
      struct elem *prec, *curr;
      /*curr sara' un puntatore all'elemento il cui campo chiave e'
      confrotntato con k*/
      /*prec sara' un puntatore all'elemento precedente a quello puntato
da    curr*/

      if (top==NULL) {/* CASO 1*/
            top=e;
            top->next=NULL;
      }
      else {
            prec=top;curr=top;
            while (curr!=NULL && curr->key <= k){
                  prec=curr;
                  curr=curr->next;
            }
            /* A questo punto in curr ci sara l'elemento che contiene un
                  campo chiave maggiore di k, oppure NULL. Mentre prec
punterà                  all'elemento precedente a quello puntato da curr*/
            if (top==curr) { /*CASO 3*/
                  e->next=curr;
                  top=e;
            }
            else { /*CASI 2 e 4 */
                  e->next=curr;
                  prec->next=e;
            }
      }
      return top;
}
```

Cancellazione di un elemento.

Vogliamo ora trovare un elemento con campo chiave pari a k in una lista concatenata ed eliminarlo. La soluzione è analoga a quella del problema precedente:

```
struct elem *trovaCancella(struct elem *top, int k) {
      struct elem *prec, *curr;
      /*curr sara' un puntatore all'elemento il cui campo chiave e'
      confrotntato con k*/
      /*prec sara' un puntatore all'elemento precedente a quello puntato
da    curr*/
      if (top!=NULL)
            curr=top;prec=top;
            while (curr!=NULL && curr->key != k){
                  prec=curr;
                  curr=curr->next;
            }
            /* A questo punto in curr ci sara l'elemento che contiene un
                  campo chiave uguale a k, oppure NULL (l'elemento non
esiste).            Mentre prec punterà  all'elemento precedente a
            quello puntato da curr*/
            if (curr!=NULL) {
                  if (top==curr) top=curr->next;
                  else  prec->next=curr->next;
                  free(curr);
            }
      }
return top;
}
```

Ordinamento di una lista concatenata

Vediamo ora come ordinare una lista tramite una versione differente della funzione inserisciInOrdine definita precedentemente. In questo caso definiamo prima una funzione che data una lista concatenata ordinata inserisce un elemento già creato nella lista data e mantenendo l'ordine (supponiamo che sia un ordine crescente):

```
struct elem *inserisciInOrdine(struct elem *top, struct elem *e) {

      struct elem *prec, *curr;
      /*curr sara' un puntatore all'elemento il cui campo chiave e'
      confrotntato con k*/
```

```
      /*prec sara' un puntatore all'elemento precedente a quello puntato
  da    curr*/
      if (e!=NULL){
            e->next=NULL;
            if (top==NULL) {/* CASO 1*/
                  top=e;
                  top->next=NULL;
            }
            else {
                  prec=top;curr=top;
                  while (curr!=NULL && curr->key <= k){
                        prec=curr;
                        curr=curr->next;
                  }
            /* A questo punto in curr ci sara l'elemento che contiene un
                  campo chiave maggiore di k, oppure NULL. Mentre prec
  punterà               all'elemento precedente a quello puntato da curr*/
                  if (top==curr) { /*CASO 3*/
                        e->next=curr;
                        top=e;
                  }
                  else { /*CASI 2 e 4 */
                        e->next=curr;
                        prec->next=e;
                  }
            }
      }
      return top;
}
```

Sulla base di tale funzione definiamo la funzione ordina:

```
  struct elem *ordina(struct elem *top) {
      struct elem *tmp;
      struct elem *nuovaLista=NULL; /*Definisco una lista vuota*/
      /*scorro la lista top*/
      while (top!=NULL) {
```

```
            /*stacco la testa della lista */
            tmp=top; top=top->next;
            /*Inserisco in ordine gli elementi di top nella nuova lista*/
            nuovaLista= insInOrdine(nuovaLista,tmp);
      }
      return nuovaLista;
}
```

Si noti che La funzione restituisce una nuova lista ordinata **distruggendo** la lista precedente.

UNA NOTA:

Le liste concatenate sono una possibile rappresentazione della struttura dati ricorsiva **lista**.

Dato un insieme di valori U, **chiamiamo lista** una sequenza finita di elementi di U definita dai seguenti tre punti:

1. Un insieme vuoto è una lista.
2. Una coppia ordinata costituita da due elementi (t,c), in cui il primo elemento t appartiene ad U ed c è una lista, è una lista. t è detta testa o top della lista. c è detto resto o coda della lista.
3. null'altro è una lista.

Esercizi

Es1. Dato una lista concatenata, scrivere una funzione iterativa che restituisca la lunghezza della lista, cioè il numero di elementi presenti nella lista.

Es2. Assegnate due liste concatenate top1 e top2 contenenti numeri interi, fondere le due liste in un'unica lista in cui nella prima parte ci siano solo i numeri dispari e nella seconda parte solo i numeri pari.

Es3. Un polinomio in una sola variabile può essere rappresentato sotto forma di lista concatenata dove ciascun elemento ha due campi (oltre il campo next): il coefficiente e la potenza della variabile (che è sempre un numero non negativo); per esempio il polinomio $P(x)= 4x^8-5x^4+8$ è rappresentato come:

(4,8) → (5,4) → (8,0) → NULL

Date, allora, due liste concatenate rappresentati due polinomi P1 e P2, rispettivamente, scrivere una funzione che restituisca una lista concatenata rappresentante la somma dei due polinomi P1 e P2.

Ad esempio:

P1(x)= $4x^8-5x^4+8$ è rappresentato come top1 → (4,8) → (-5,4) → (8,0) → NULL

P2(x)= $2x^6+3x^4-2$ è rappresentato come top2 → (2,6) → (3,4) → (-2,0) → NULL

Si deve ottenere:

P1(x) +P2(x)= $4x^8+2x^6-2x^4+6$ rappresentato come top → (4,8) → (2,6) → (-2,4) → (6,0) → NULL

Es4. Siano L1 ed L2 due liste concatenate che contengono interi, con L1 che contiene anche ripetizioni. Costruire una terza lista L3, senza ripetizioni, che contenga tutti gli elementi di L1 non appartenenti ad L2 .

Es5. Trasformare una lista concatenata L disordinata e contenente ripetizioni in una lista ordinata senza ripetizioni che contiene soltanto gli elementi di L che si ripetono esattamente k volte.