

Unità Didattica 6

# Linguaggio C

Strutture dati lineari:  
Liste lineari, Stack e Code

# Liste Lineari

- Insieme ordinato di elementi di uno stesso tipo, tali che ogni elemento ha un predecessore e un successore, ad esclusione del primo e dell'ultimo;
  - 1)  $x_0$  è il primo elemento della lista ed è seguito da  $x_1$ ;
  - 2)  $x_i$  è preceduto da  $x_{i-1}$  e seguito da  $x_{i+1}$ ,  $\forall 1 \leq i \leq n-2$ ;
  - 3)  $x_{n-1}$  è l'ultimo elemento della lista ed è preceduto da  $x_{n-2}$ ;
- $n$  è la lunghezza della lista; se  $n = 0$  la lista è vuota;
- La relazione di precedenza significa che per accedere ad un elemento bisogna accedere a tutti quelli che lo precedono;

# Liste e Vettori

- Rispetto ai vettori, le liste sono caratterizzate da:
  - 1) Numero variabile di elementi (strutture dati dinamiche (come strutture astratte));
  - 2) Modalità di accesso ai dati (in un vettore si può accedere direttamente al dato, in una lista per accedere ad un dato occorre prima accedere al precedente).

# Realizzazioni concrete di liste: mediante vettori di strutture

- Si possono realizzare con vettori di strutture di tipo:

```
struct nodo
{
    tipoinfo info;
    int next;
};
struct nodo vett_lista[DIM];
```

vett\_lista      info      next

0	info0	3
1	info1	4
2	info2	...
3	info3	1
	...	...
DIM-1		

Problemi nella realizzazione tramite vettori:

- 1) Spreco di memoria a causa della dimensione ignota delle liste;
- 2) Uso di una struttura che consente l'accesso diretto ai dati (improprio per le liste).

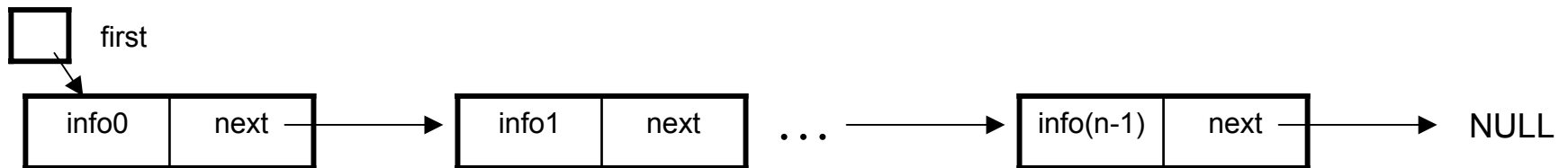
vett\_lista[i].next  
contiene l'indice dell'elemento  
successivo all'elemento  
vett\_lista[i]

# Realizzazioni concrete di liste: mediante strutture concatenate

- Il campo next è rappresentato da un “puntatore” ad una struttura nodo della lista:

```
struct nodo
{
    tipoinfo info;
    struct nodo *next;
};
```

- Si vuole allocare complessivamente una struttura dati con la seguente configurazione:



# Lista con strutture concatenate con dati letti da file

```
...
struct studente
{
    char nome[30];
    char cognome[50];
    int voto_informatica;
    struct studente *next;
};
typedef struct studente *p_st;
p_st stud, stud_last, stud_first;

if((fp = fopen("file.dat", "r")) == NULL) exit(0);
stud_first = stud_last = NULL;
while(!feof(fp))
{
    stud = malloc(sizeof(struct studente));
    stud->next = NULL;
    fscanf(fp, "%s %s %d\n", stud->nome, stud->cognome,
           &(stud->voto_informatica));
    if (stud_first == NULL)
        stud_first = stud;
    else
        stud_last->next = stud;

    stud_last = stud;

    fscanf(fp, "\n");
}
...
```

file.dat

```
Mario Rossi 25
Giorgio Bianchi 30
... ..
```

# Visualizzazione elementi lista

...

```
stud = stud_first; //stud_first contiene l'indirizzo al primo elemento della
                  //lista
while(stud != NULL)
{
    printf("%20s ", stud->nome);

    printf("%20s ", stud->cognome);

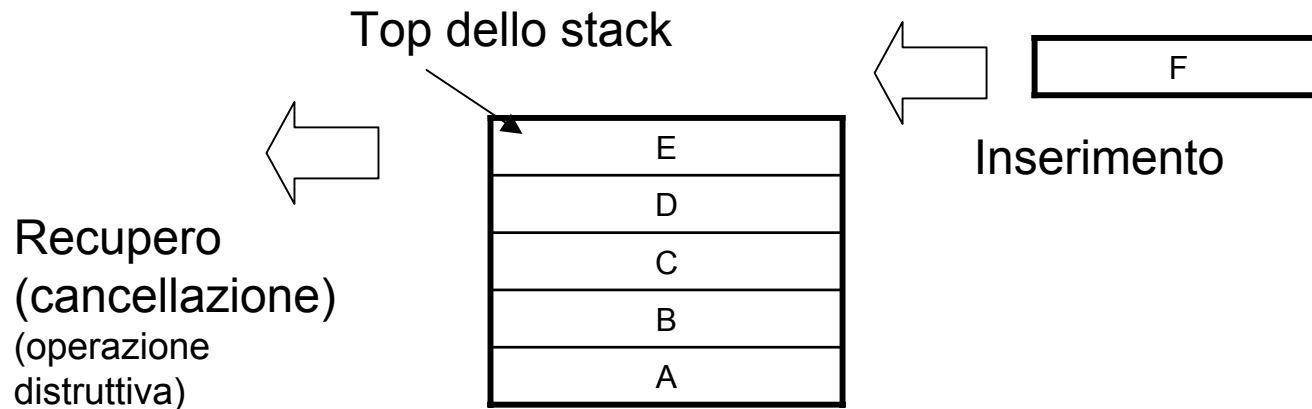
    printf("%2d\n", stud->voto_informatica);

    stud = stud->next;
}
```

...

# Stack (o Pila)

- Lista lineare in cui gli inserimenti e le estrazioni avvengono da un estremo detto testa o top dello stack;
- Politica di gestione LIFO (Last Input First Output);
- Si può realizzare con vettori o con liste concatenate.



# Realizzazione di stack mediante strutture concatenate.

## Alg. di inserimento (push) e estrazione (pop)

```
void push(tipoinfo x)
{
    static struct nodo *first_elem;
    struct nodo *elem;

    elem = (struct nodo *)
        malloc(sizeof(struct nodo));
    elem->info = x;
    elem->next = NULL;

    if (first_elem == NULL)
    {
        first_elem = elem;
    }
    else
    {
        elem->next = first_elem;
        first_elem = elem;
    }
}
```

```
struct nodo *pop()
{
    static struct nodo *first_elem;
    struct nodo *elem;

    if (first_elem == NULL)
        return NULL;

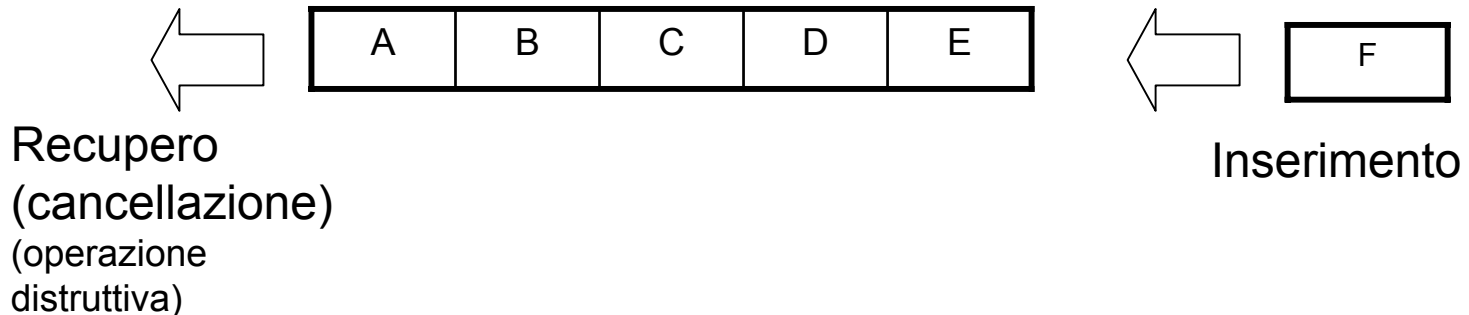
    elem = first_elem;

    first_elem = first_elem->next;

    return(elem);
}
```

# Coda

- Lista lineare in cui le estrazioni avvengono da un lato e gli inserimenti dall'altro;
- Politica di gestione FIFO (First Input First Output);
- Si può realizzare con un vettore o con liste di oggetti concatenati.



# Realizzazione di code mediante strutture concatenate.

## Alg. di inserimento (push) e estrazione (pop)

```
void push(tipoinfo x)
{
    static struct nodo *first_elem,*last_elem;
    struct nodo *elem;

    elem = (struct nodo *)
        malloc(sizeof(struct nodo));
    elem->info = x;
    elem->next = NULL;

    if (last_elem == NULL)
    {
        last_elem = elem;
        first_elem = elem;
    }
    else
    {
        last_elem->next = elem;
        last_elem = last_elem->next;
    }
}
```

```
struct nodo *pop()
{
    static struct nodo *first_elem;
    struct nodo *elem;

    if (first_elem == NULL)
        return NULL;

    elem = first_elem;

    first_elem = first_elem->next;

    return(elem);
}
```

# Proprietà di liste, stack e code

- Stack e code si possono realizzare anche mediante vettori in cui ogni elemento contiene solo l'informazione, essendo note le posizioni di inserimento e di cancellazione;
- Nelle liste, inserimenti e cancellazioni possono essere effettuati in qualunque posizione e determinano una modifica dei riferimenti fra i nodi.