

Unità didattica: Particolari organizzazioni dei dati per una lista lineare [7-AC]

Titolo: Operazioni sulle liste lineari con nodo “sentinella”
e sulle liste lineari “generiche”

Argomenti trattati:

- ✓ Come eliminare le differenze tra l’inserimento e l’eliminazione di un nodo alla testa ed nel mezzo della lista
- ✓ Puntatori a “void” e “cast”
- ✓ Nuova organizzazione dei dati
- ✓ Nuovi prototipi delle function C per le operazioni sulla struttura dati
- ✓ Nuova chiamata alle function

Prerequisiti richiesti: implementazione C di una lista lineare

Per evitare di avere due funzioni per l'inserimento e due per l'eliminazione, per la necessità di gestire diversamente il puntatore al nodo di testa e quello ad un nodo generico, si può inserire all'inizio della lista un nodo fittizio (detto *nodo sentinella*).



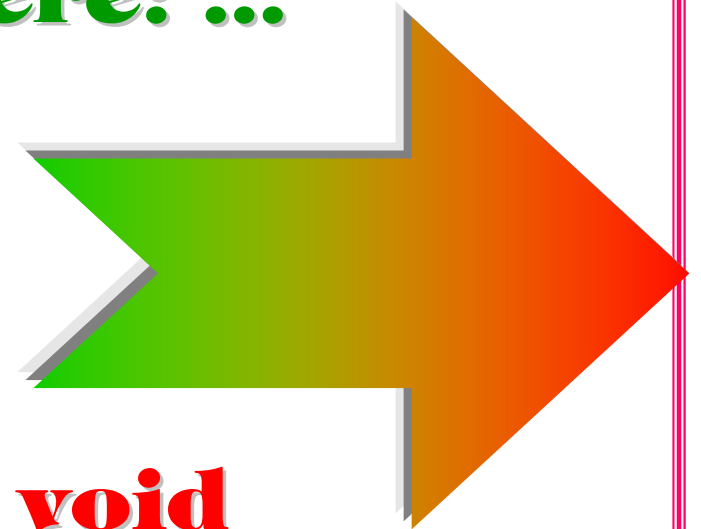
È possibile scrivere delle funzioni in C (come le precedenti **crea_lista()**, **insl_testa()**, **insl_nodo()**, ...) che implementino le operazioni sulla lista **senza far riferimento alla particolare struttura** (**struct PERSONA**) definita per il nodo della lista nel main ?

SI !!! **Bisogna ricorrere: ...**

... al cast

... all'uso

dei puntatori a void



In C: operazioni sulle liste generiche

08_07.4



testa

**organizzazione
dei dati**

```
typedef struct{
```

tipo del campo informa-
zione del nodo della lista
(globale)

```
char nome[20];  
short eta;  
} INFO_FIELD;
```

...

```
void main()
```

**struttura della lista
(locale nel main)**

```
{struct PERSONA
```

```
{INFO_FIELD info;  
struct PERSONA *p_next;};
```

**esempio
di dati**

```
struct PERSONA *head, *punt;
```

```
char *name[]={ "Bianchi Roberto",...};  
short age[]={22,25,18,...};
```

prototipo delle funzioni

nuovo parametro

puntatore

```
void *creaLista()  
void insL_testa( short , INFO_FIELD *, void ** )  
void insL_nodo ( short , INFO_FIELD *, void ** )
```

```
/* crea lista vuota */  
void *creaLista()  
{char *testa;  
  testa=NULL;  
  return testa;  
}
```

puntatori
generici


cast

chiamata nel main():

```
head = (struct PERSONA *) creaLista();
```

```
/* inserisce dato in testa alla lista */  
void insL_testa(short len_info,  
                INFO_FIELD *p_dato, void **p_head)  
{  
    struct lista  
        {INFO_FIELD info;  
          struct lista *p_next;  
        } *ptr;  
    ptr=calloc(1, sizeof(struct lista));  
    memcpy(ptr, p_dato, len_info);  
    ptr->p_next=(struct lista *)*p_head;  
    *p_head = ptr;  
}
```

tipo generico
lista lineare



chiamata:

```
insL_testa(len_info, p_nuovodato, (void **)&head);
```

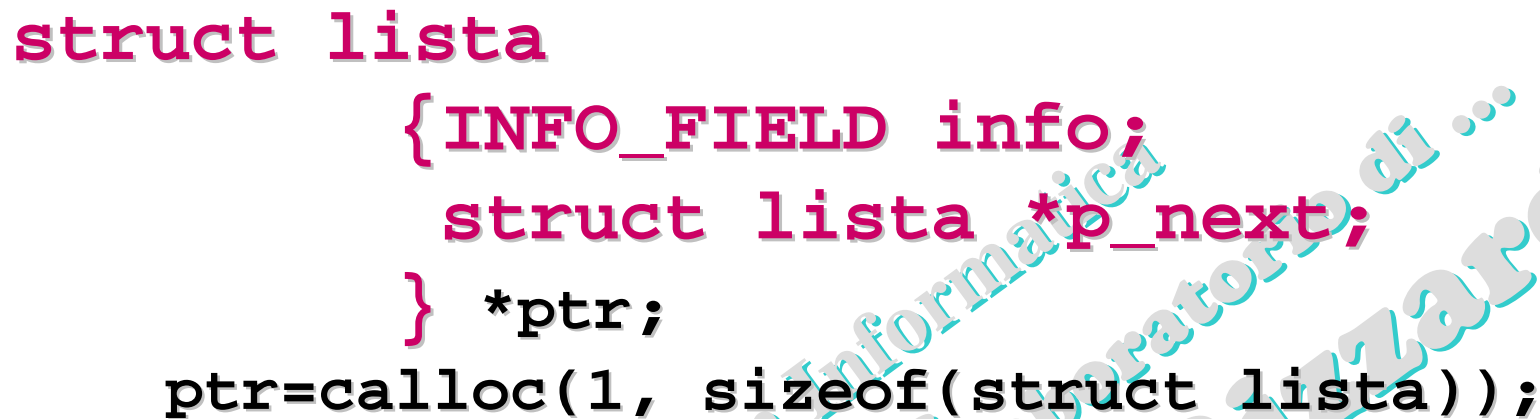
```
/* inserisce dato dopo nodo corrente */
void insL_nodo(short len_info,
               INFO_FIELD *p_dato, void **p_punt)
{
    struct lista
    {
        INFO_FIELD info;
        struct lista *p_next;
    } *ptr;
    ptr=calloc(1, sizeof(struct lista));
    memcpy(ptr, p_dato, len_info);
    ptr->p_next=((struct lista *)*p_punt)->p_next;
    ((struct lista *)*p_punt)->p_next=ptr;
    *p_punt=ptr;
}
```

**tipo generico
lista lineare**

chiamata:

```
insL_nodo(len_info, p_nuovodato, (void **)&punt);
```

```
struct lista
{
    INFO_FIELD info;
    struct lista *p_next;
} *ptr;
ptr=calloc(1, sizeof(struct lista));
```



definisce una struttura lista generica che condivide con il main solo il tipo INFO_FIELD (globale).

```
memcpy(ptr, p_dato, len_info);
```



copia len_info byte da *p_dato a *ptr senza interessarsi della struttura delle informazioni


```
ptr->p_next=(struct lista *)*p_head;  
*p_head=ptr;
```

Il cast `(struct lista *)` è **obbligatorio** prima di usare un puntatore generico, ma non ci vuole quando lo si definisce.

```
ptr->p_next =  
    ((struct lista *)*p_punt)->p_next;  
((struct lista *)*p_punt)->p_next=ptr;  
*p_punt=ptr;
```

Analogamente per l'eliminazione di un nodo

```
/* elimina nodo in testa alla lista */  
void eliL_testa(void **p_head)  
{  
    struct lista  
    {  
        INFO_FIELD info;  
        struct lista *p_next;  
    } *ptr;  
    ptr=((struct lista *)*p_head)->p_next;  
  
    free((struct lista *)*p_head);  
    *p_head=ptr;  
}
```

dealloca la memoria
puntata da *p_head.

chiamata:

```
eliL_testa((void **)&head);
```

```

/* elimina nodo successore */
void eliL_nodo(void **p_punt)
{
    struct lista
    {
        INFO_FIELD info;
        struct lista *p_next;
    } *ptr;
    ptr=((struct lista *)*p_punt)->p_next;
    ((struct lista *)*p_punt)->p_next =
        ptr->p_next;
    free(ptr);
}

```

chiamata:

```
eliL_testa((void **)&prec);
```

dove **prec** punta al nodo che precede quello da eliminare.