

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*).



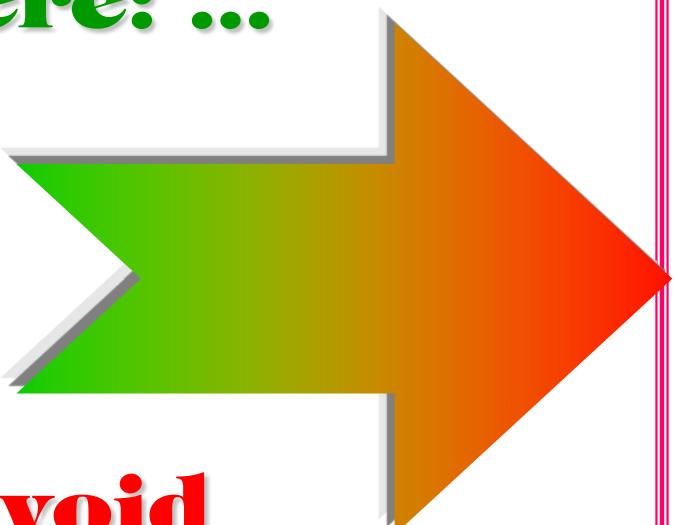
È possibile scrivere delle funzioni in *C* (come le precedenti **crea_list()**, **insl_testa()**, **insl_nodo()**, ...) che implementino le operazioni sulla lista **senza far riferimento alla particolare struttura (**struct PERSONA**)** definita per il nodo della lista nel main ?

SI !!! Bisogna ricorrere: ...

... al **cast**

... all'uso

dei puntatori a void



In C: operazioni sulle liste generate

08_07.4



testa

typedef struct{

tipo del campo informazione del nodo della lista (globale)

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

...

void main()

{ struct PERSONA

struttura della lista (locale nel main)

{ INFO_FIELD info;

struct PERSONA *p_next; } ;

struct PERSONA *head, *punt;

char *name[]={"Bianchi Roberto",...};

short age[]={22,25,18,...};

esempio di dati

organizzazione dei dati

Laura Strutture dinamiche lineari

(prof. M. Rizzardi)

prototipo delle funzioni

```
void *creaLista()  
void insL_testa(  
void insL_nodo (
```

nuovo parametro

short
short

INFO_FIELD *,
INFO_FIELD *,

puntatore dato

void **)
void **)

cast

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

puntatori generici

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;
```

**tipo generico
lista lineare**

```
ptr=malloc(1, sizeof(struct lista));
memcpy(&(ptr->info), p_dato, len_info);
ptr->p_next=(struct lista *)*p_head;
*p_head = ptr;
}
```

```
void *memcpy(void *destination, const void *source, size_t num);
destination is returned
```

chiamata:

```
insL_testa(sizeof(INFO_FIELD), p_nuovodato, (void **)&head);
```

```

/* inserisce dato dopo nodo corrente e aggiorna p_punt */
void insL_nodo(short len_info,
                INFO_FIELD *p_dato, void **p_punt)
{
    struct lista
    {
        INFO_FIELD info;
        struct lista *p_next;
    } *ptr;
    ptr=malloc(1, sizeof(struct lista));
    memcpy(&(ptr->info), p_dato, len_info);
    ptr->p_next=((struct lista *)*p_punt)->p_next;
    ((struct lista *)*p_punt)->p_next=ptr;
    *p_punt=ptr; /* fa avanzare il puntatore */
}

```

**tipo generico
lista lineare**

chiamata:

```
insL_nodo(sizeof(INFO_FIELD), p_nuovodato, (void **)&punt);
```

```
struct lista
```

```
{INFO_FIELD info;  
 struct lista *p_next;  
} *ptr;
```

```
ptr=(struct lista*)calloc(1, sizeof(struct lista));
```

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

```
memcpy(&(ptr->info), p_dato, len_info);
```

copia len_info byte da *p_dato a *(ptr->info) 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 di p_punt */
void eliL_nodo(void *p_punt)
{
    struct lista
    {   INFO_FIELD info;
        struct lista *p_next;
    } *ptr;
    if (p_punt == NULL) return; /* non c'è nulla da eliminare */
    ptr=((struct lista *)p_punt)->p_next; /* punta al successore */
    ((struct lista *)p_punt)->p_next = ptr->p_next;
    free(ptr);
}
```

chiamata:

```
eliL_nodo((void *)prec);
```

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