

Esercitazione di Reti degli elaboratori

Prof.ssa Chiara Petrioli



SAPIENZA
UNIVERSITÀ DI ROMA

Corso di C

Christian Cardia, Gabriele Saturni

Cosa vedremo in questa lezione?

- **Struct**
- **Typedef**
- **Union**
- **Liste**
- **Esercizi**



Tipi di dati strutturati

Struct

- Permette di creare tipi di dati strutturati
- L'obiettivo è quello di creare una struttura che contiene più variabili di differente tipo
- Per definirla è opportuno utilizzare la parola chiave **struct** seguita dal nome che identifica la struttura stessa (il nome che la identifica è opzionale)

Struct

Dichiarazione

```
struct <nome_tipo> {  
  <tipo_campo_1> <nome_campo_1>;  
  <tipo_campo_2> <nome_campo_2>;  
  ....  
  ....  
  <tipo_campo_n> <nome_campo_n>;  
} <nome_var_1>, ..., <nome_var_2>;
```

- Viene definito un nuovo tipo

struct <nome_tipo>
contenente le variabili
definite al suo interno.

Struct

Dichiarazione, inizializzazione e accesso ai campi

- Dichiarazione di un'istanza di tipo *nome_tipo*

`struct <nome_tipo> <nome_variabile>;`

- Accesso agli elementi
- (utilizzare operatore ".")

`<nome_variabile>.<nome_campo>`

- Dichiarazione e inizializzazione
(come per gli array)

`struct <nome_tipo> <nome_variabile> = {<valore_campo_1>, ..., <valore_campo_n> };`

Struct

Un esempio

```
3 struct Articolo {  
4     int codice;  
5     float prezzo;  
6 };  
7  
8 int main(){  
9     //dichiarazione e inizializzazione  
10    struct Articolo art1 = { 467, 12.5};  
11  
12    //dichiarazione  
13    struct Articolo art2;  
14    //accesso ai campi in scrittura  
15    art2.codice = 189;  
16    art2.prezzo = 34.99;  
17  
18    //accesso ai campi in lettura  
19    printf("art1 - codice: %d, prezzo: %0.2f \n", art1.codice, art1.prezzo);  
20    printf("art2 - codice: %d, prezzo: %0.2f \n", art2.codice, art2.prezzo);
```

Struct

- È possibile dichiarare un puntatore ad una *struct*:

struct <nome_tipo> *<nome_puntatore>;

- Assegnare al puntatore una struttura creata:

<nome_puntatore> = &<nome_variabile>;

- 
- Indirizzo in memoria della struttura <nome_variabile>

<nome_puntatore> -> <nome_variabile>

- 
- Operatore per accedere ai campi del puntatore

Struct e puntatori

Un esempio (1/2)

```
3 struct Articolo {  
4     int codice;  
5     float prezzo;  
6 };  
7 int main(){  
8     struct Articolo art = { 1234, 12.99 };  
9     printf("codice:%d,prezzo:%0.2f \n"  
10    ,art.codice,art.prezzo);  
11  
12     struct Articolo *p;  
13     p = &art;  
14     p->codice = 567;  
15     p->prezzo = 3.99;  
16  
17     printf("codice:%d,prezzo:%0.2f \n"  
18    ,art.codice,art.prezzo);
```


Struct e puntatori

Un esempio (2/2)

```
3 struct Articolo {  
4     int codice;  
5     float prezzo;  
6 };  
7 int main(){  
8     struct Articolo art = { 1234, 12.99 };  
9     printf("codice:%d,prezzo:%0.2f \n"  
10    ,art.codice,art.prezzo);  
11
```

art (Articolo)

- codice=1234
- prezzo=12.99

```
12 struct Articolo *p;  
13 p = &art;  
14 p->codice = 567;  
15 p->prezzo = 3.99;  
16  
17 printf("codice:%d,prezzo:%0.2f \n"  
18    ,art.codice,art.prezzo);
```

p

art (Articolo)

- codice=567
- prezzo=3.99

Struct

Un altro esempio...

```
3 struct Articolo {  
4     int codice;  
5     float prezzo;  
6 }art2;  
7  
8 int main(){  
9  
10     struct Articolo art = { 1234, 12.99, "articolo" };  
11     printf("art - codice:%d,prezzo:%0.2f \n",  
12         art.codice,art.prezzo);  
13  
14     art2.codice = 6789;  
15     art2.prezzo = 2.99;  
16     printf("art2 - codice:%d,prezzo:%0.2f \n",  
17         art2.codice,art2.prezzo);
```

- È possibile creare direttamente una variabile di tipo Articolo e di nome *art2*

- Creo e inizializzo un'altra **struct** di tipo Articolo e di nome *art*

- Inizializzo gli elementi di *art2* (creato all'inizio)

Struct

Esercizio 1

- Si dichiara una struttura di nome *Automobile* che contiene i seguenti campi: *prezzo*, *modello*, *cilindrata*, *colore*.
- Il programma deve permettere all'utente di salvare tre tipi di macchine differenti (quindi deve poter inserire in input tutti i campi delle tre rispettive macchine)
- Infine, si stampino tutti i campi delle tre macchine

Esercizio 1

Soluzione (1/2)

```
3 struct Automobile{
4     float prezzo;
5     char modello[20];
6     int cilindrata;
7     char colore[20];
8 }auto1,auto2,auto3;
9
10 int main(){
11     printf("Auto1-\nPrezzo: ");
12     scanf("%f",&auto1.prezzo);
13     printf("\nModello: ");
14     scanf("%s",&auto1.modello);
15     printf("\nCilindrata: ");
16     scanf("%d",&auto1.cilindrata);
17     printf("\nColore: ");
18     scanf("%s",&auto1.colore);
19     printf("\n\n");//-----
20     printf("Auto2-\nPrezzo: ");
21     scanf("%f",&auto2.prezzo);
22     printf("\nModello: ");
23     scanf("%s",&auto2.modello);
24     printf("\nCilindrata: ");
25     scanf("%d",&auto2.cilindrata);
26     printf("\nColore: ");
27     scanf("%s",&auto2.colore);
28     printf("\n\n");//-----
```

- Dichiarazione delle tre strutture *auto1*, *auto2*, *auto3* di tipo *Automobile*

- Inserimento input della struttura *auto1*

- Inserimento input della struttura *auto2*

Esercizio 1

Soluzione (2/2)

```
29 printf("Auto3-\nPrezzo: ");
30 scanf("%f",&auto3.prezzo);
31 printf("\nModello: ");
32 scanf("%s",&auto3.modello);
33 printf("\nCilindrata: ");
34 scanf("%d",&auto3.cilindrata);
35 printf("\nColore: ");
36 scanf("%s",&auto3.colore);
37
```

- Inserimento input della struttura *auto3*

```
38 printf("\n\nAuto inserite:\n");
39 printf("Auto1- Prezzo: %f, Modello: %s, Cilindrata: %d, Colore: %s",
40 auto1.prezzo,auto1.modello,auto1.cilindrata,auto1.colore);
41 printf("\nAuto2- Prezzo: %f, Modello: %s, Cilindrata: %d, Colore: %s",
42 auto2.prezzo,auto2.modello,auto2.cilindrata,auto2.colore);
43 printf("\nAuto3- Prezzo: %f, Modello: %s, Cilindrata: %d, Colore: %s",
44 auto3.prezzo,auto3.modello,auto3.cilindrata,auto3.colore);
45
```

- Stampa dei campi delle tre strutture Automobile

Typedef

- Permette di fornire a un tipo un nuovo nome
- Può essere utilizzato per fornire un nome alla nostra struttura e quindi definire un nuovo tipo di dato con quel nome
- In questo modo si evita l'uso della parola chiave **struct**
- La sintassi è quella per dichiarare una variabile ma preceduta dalla parola chiave **typedef**

typedef <tipo> <nome>;

Esempio con **struct**:

```
2 ▢ typedef struct Libri{
3     int numPagine;
4     char titolo[50];
5     float prezzo;
6 } libro;
7
8 ▢ int main(){
9
10     libro libro1 = {354, "Fondamenti di programmazione", 42.50};
```

- Ora *libro* è diventato un tipo (in particolare una struttura di tipo *Libri*)

- Quindi è possibile dichiarare una variabile di tipo *libro* (*libro* è diventato un tipo di dato)

Esercizio 2

- Si scriva un programma che dichiari una struttura di nome Libro con due campi: *titolo* e *prezzo*
- Si dichiari una funzione che prende in input due libri e ritorna 0 se hanno gli stessi valori nei due rispettivi campi, 1 altrimenti
- N.B. i valori dei campi delle strutture devono essere inseriti in input dall'utente

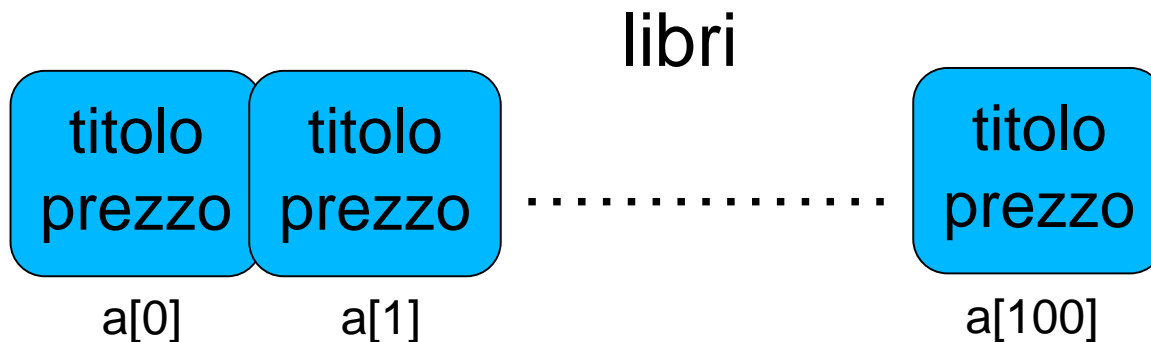
Esercizio 2 Soluzione

```
3 typedef struct Libro{
4     char titolo[50];
5     float prezzo;
6 }Libro;
7 int controlloLibri(Libro l1, Libro l2);
8 int main(){
9     Libro libro1,libro2;
10    printf("Libro1-->");
11    printf("\ntitolo: ");
12    scanf("%s",&libro1.titolo);
13    printf("\nprezzo: ");
14    scanf("%f",&libro1.prezzo);
15    printf("Libro2-->");
16    printf("\ntitolo: ");
17    scanf("%s",&libro2.titolo);
18    printf("\nprezzo: ");
19    scanf("%f",&libro2.prezzo);
20    if(controlloLibri(libro1,libro2)==0) printf("I libri sono uguali!");
21    else printf("I libri sono diversi!");
22    return 0;
23 }
24 int controlloLibri(Libro l1, Libro l2){
25     if ( (l1.prezzo==l2.prezzo) && (strcmp(l1.titolo,l2.titolo)==0) ){
26         return 0;
27     }
28     return 1;
29 }
```


Array di Struct

Dichiarazione

- Come per i tipi di dato primitivi, è possibile dichiarare array di **struct**
- In merito all'esercizio precedente, possiamo dichiarare un Array di Libri nel seguente modo:
- *Libro libri[100];*
- Con questa istruzione abbiamo dichiarato un array di 100 elementi, ognuno dei quali è una struttura di tipo Libro

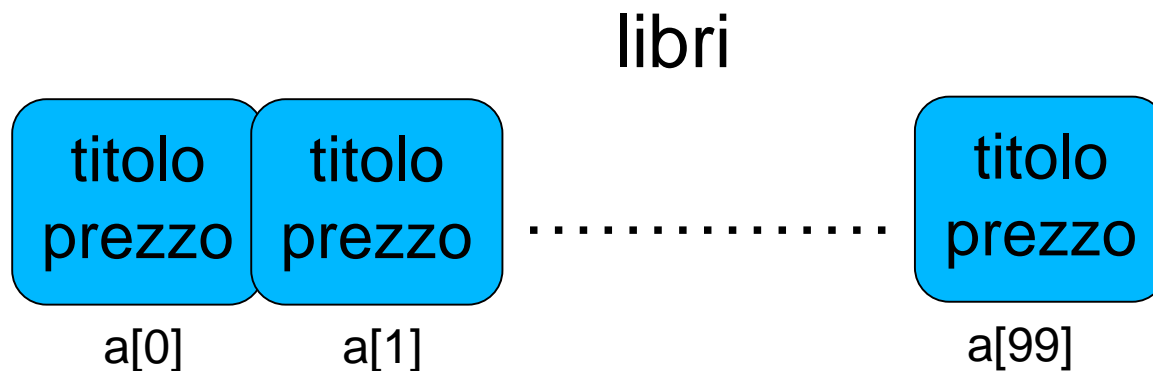


- L'accesso è lo stesso come con gli array di tipi primitivi

Array di Struct

Accedere ai campi

- Accedere ai campi della struttura alla posizione i , tale che $0 \leq i < 100$
- `libri[i].prezzo = 12.54;`
- `printf ("%f" , libri[i].prezzo);`



Union

- Tipo di dato speciale che permette di salvare differenti tipi di dato **nella stessa locazione di memoria**
- È possibile definire un tipo **union** con molti membri, ma in ogni istante un solo membro può contenere un valore
- È un metodo per permettere un efficiente utilizzo della stessa zona di memoria
- La memoria occupata è grande abbastanza da contenere il più grande membro definito nella **union**
- Definizione:

```
union <tag> {  
    <tipo_1> <nome_1>;  
    ...  
    <tipo_n> <nome_n>;  
} <una o più variabili union>;
```

N.B. Il modo di accedere ai membri è uguale a quello delle **struct**

Union

Esempio

```
3 union Dato {  
4     int i;  
5     char c;  
6 } dato;  
7  
8 int main(){  
9     printf( "Memoria occupata: %d\n", sizeof(dato) );  
10    return 0;  
11 }
```



Che cosa stampa la
funzione *printf* ?

Union

Esempio



- Viene allocato spazio per contenere la variabile più grande
- un int occupa 4 bytes, un char solo 1 byte, quindi...
- La union *dato* occuperà 4 bytes

```
3 union Dato {  
4     int i;  
5     char c;  
6 } dato;  
7  
8 int main(){  
9     printf( "Memoria occupata: %d\n", sizeof(dato) );  
10    return 0;  
11 }
```

Union

Un altro esempio

```
4 union Dato {  
5     int i;  
6     float f;  
7     char str[20];  
8 };  
9  
10 int main( ) {  
11  
12     union Dato dato;  
13  
14     dato.i = 34;  
15     dato.f = 12.37;  
16     strcpy( dato.str, "Programmazione C");  
17  
18     printf( "dato.i : %d\n", dato.i);  
19     printf( "dato.f : %f\n", dato.f);  
20     printf( "dato.str : %s\n", dato.str);
```



Che cosa stampano
le tre funzioni *printf* ?

Union

Un altro esempio

```
4 union Dato {  
5     int i;  
6     float f;  
7     char str[20];  
8 };
```

Condividono la stessa zona di memoria di 20 bytes (20 bytes in quanto è la dimensione del tipo più grande, ovvero str)

```
10 int main( ) {  
11  
12     union Dato dato;  
13  
14     dato.i = 34;  
15     dato.f = 12.37;  
16     strcpy( dato.str, "Programmazione C");  
17  
18     printf( "dato.i : %d\n", dato.i);  
19     printf( "dato.f : %f\n", dato.f);  
20     printf( "dato.str : %s\n", dato.str);
```

L'assegnazione dell'ultima, esclude le altre
(condividono lo stesso spazio in memoria)

```
dato.i : 1735357008  
dato.f : 11307542828377711000000000.000000  
dato.str : Programmazione C
```

Esercizio 3

- Si scriva un programma che dichiari una struttura di nome *Persona* con i seguenti campi: *nome*, *cognome*, *eta*, *dataNascita*
- N.B. *dataNascita* deve essere un'altra struttura composta dai campi *giorno*, *mese* e *anno*.
- Si crei un Array di cinque posizioni di tipo *Persona*
- Permettere all'utente di inserire in input i dati di tutte le persone
- Infine stampare tutti i dati inseriti

Esercizio 3 Soluzione (1/2)

```
5 typedef struct Data {
6     int giorno;
7     int mese;
8     int anno;
9 }data;
10
11 typedef struct Persona{
12     char nome[20];
13     char cognome[20];
14     int eta;
15     data dataNascita;
16 }persona;
17
18 int main(){
19     persona persone[5];
20     int i=0;
21     for(i=0;i<5;i++){
22         printf("\nPersona %d",i+1);
23         printf("\nNome: ");
24         scanf("%s",&persone[i].nome);
25         printf("\nCognome: ");
26         scanf("%s",&persone[i].cognome);
27         printf("\nEta: ");
28         scanf("%s",&persone[i].eta);
29         printf("\nData nascita (gg/mm/aa): ");
30         scanf("%d/%d/%d",
31             &persone[i].dataNascita.giorno,&persone[i].dataNascita.mese,&persone[i].dataNascita.anno);
32     }
33     printf("\n\nLista Persone: \n");
34     for(i=0;i<5;i++){
35         printf("\nPersona %d",i+1);
36         printf("\nNome: %s, Cognome: %s, Eta: %d, Data Nascita: %d/%d/%d",
37             persone[i].nome,persone[i].cognome,persone[i].eta,
38             persone[i].dataNascita.giorno,persone[i].dataNascita.mese,persone[i].dataNascita.anno);
39     }
```

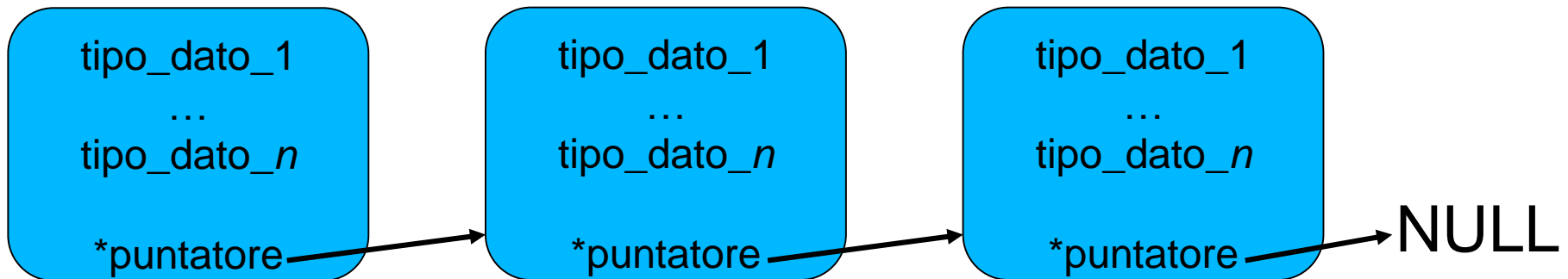
Esercizio 3 Soluzione (2/2)

```
5 typedef struct Data {
6     int giorno;
7     int mese;
8     int anno;
9 }data;
10
11 typedef struct Persona{
12     char nome[20];
13     char cognome[20];
14     int eta;
15     data dataNascita;
16 }persona;
17
18 int main(){
19     persona persone[5];
20     int i=0;
21     for(i=0;i<5;i++){
22         printf("\nPersona %d",i+1);
23         printf("\nNome: ");
24         scanf("%s",&persone[i].nome);
25         printf("\nCognome: ");
26         scanf("%s",&persone[i].cognome);
27         printf("\nEta: ");
28         scanf("%s",&persone[i].eta);
29         printf("\nData nascita (gg/mm/aa): ");
30         scanf("%d/%d/%d",
31             &persone[i].dataNascita.giorno,&persone[i].dataNascita.mese,&persone[i].dataNascita.anno);
32     }
33     printf("\n\nLista Persone: \n");
34     for(i=0;i<5;i++){
35         printf("\nPersona %d",i+1);
36         printf("\nNome: %s, Cognome: %s, Eta: %d, Data Nascita: %d/%d/%d",
37             persone[i].nome,persone[i].cognome,persone[i].eta,
38             persone[i].dataNascita.giorno,persone[i].dataNascita.mese,persone[i].dataNascita.anno);
39     }
```

Dichiarazione delle due strutture.
N.B. La struttura *data* viene usata dalla struttura *persona* per dichiarare una data di nascita.

Liste

- Una lista è una collezione di dati omogenei
- A differenza degli array, occupa in memoria una posizione qualsiasi
- La dimensione non è nota a priori e può cambiare
- Ogni elemento della lista può contenere uno o più campi e **deve necessariamente contenere un puntatore al prossimo elemento**



- L'ultimo elemento ha un puntatore a **NULL**, per indicare la fine della lista
- **È fondamentale avere un puntatore al primo elemento della lista che non deve mai essere perso!**

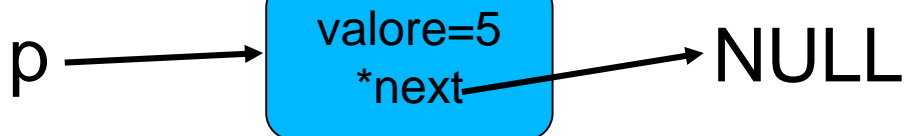
Liste

Creazione...

```
5 struct Elemento {  
6     int valore;  
7     struct Elemento *next;  
8 };  
9  
10 int main(){  
11     struct Elemento *p = (struct Elemento*) malloc(sizeof(struct Elemento));  
12     p->valore = 5;  
13     p->next = NULL;
```

Allocazione dinamica di
un nuovo Elemento

Operatore per accedere ai campi
di una struct, tramite un puntatore



Liste

Creazione...

```
p->next = (struct Elemento*) malloc(sizeof(struct Elemento));  
p->next->valore = 8;  
p->next->next = NULL;
```

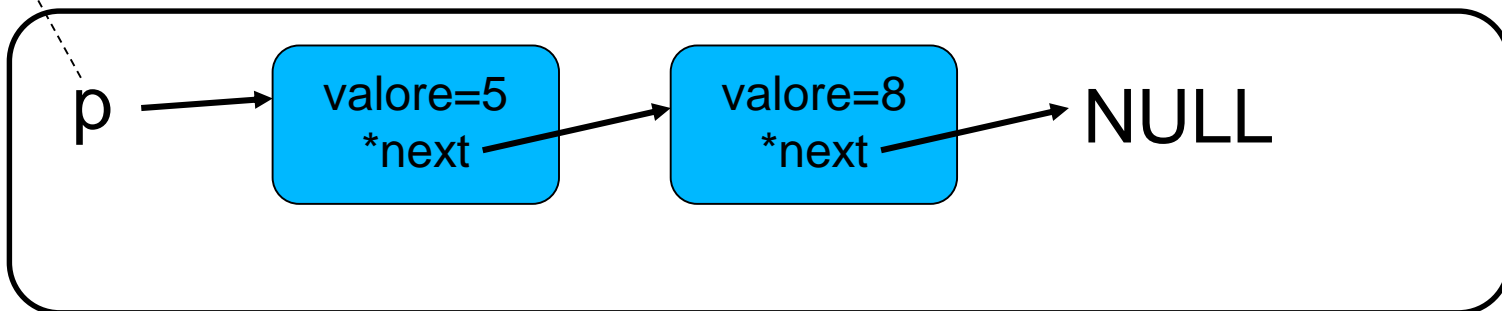
**next* punta al prossimo elemento

Allocato spazio per una struttura di tipo Elemento

Viene impostato il campo *valore* del secondo elemento a 8

Non viene MAI modificato.
Punta al primo elemento della lista!!

Il campo *next* del secondo elemento viene impostato a NULL (*fine della lista*)

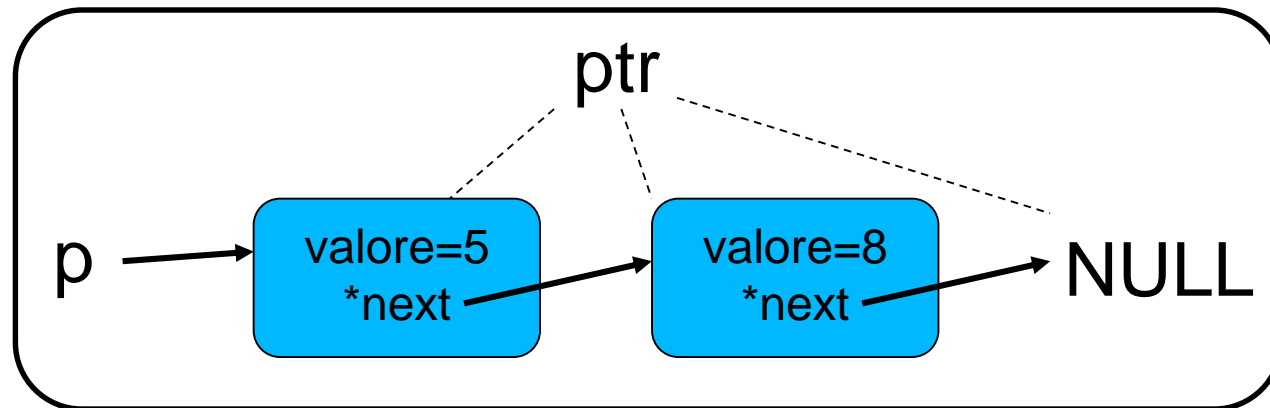


Liste Scorrimento...

```
19 struct Elemento *ptr = p;  
20  
21 while(ptr!=NULL){  
22     printf("-->%d", ptr->valore);  
23     ptr = ptr->next;  
24 }
```

Output: -->5-->8

- Si dichiara un nuovo puntatore Elemento che punta al primo elemento della lista
- Poi si fa scorrere *ptr* e stampare tutti gli elementi
- **Non modificare il puntatore al primo elemento della lista!**



Liste.....esempio

Creazione e stampa di una semplice lista

```
5 struct Elemento {  
6     int valore;  
7     struct Elemento *next;  
8 };  
9  
10 struct Elemento *creaLista();  
11 void stampaLista(struct Elemento *puntatore);  
12  
13 int main(){  
14  
15     struct Elemento *primoElemento;  
16  
17     primoElemento = creaLista();  
18  
19     stampaLista(primoElemento);  
20  
21 }
```

- Dichiarazione di una *struct* Elemento
- Prototipi di due funzioni
- *Main* che richiama le due funzioni per creare e stampare una lista

```

32 struct Elemento *creaLista(){
33
34     struct Elemento *puntatoreInizio, *puntatore;
35     int i, numElementi;
36     printf("Inserire numero elementi: ");
37     scanf("%d",&numElementi);
38
39     if(numElementi==0) puntatoreInizio == NULL;
40     else {
41         puntatoreInizio = (struct Elemento*) malloc(sizeof(struct Elemento));
42         if(puntatoreInizio==NULL){
43             printf("\nNon è possibile allocare spazio! Lista vuota...");
44             return puntatoreInizio;
45         }
46         printf("Inserisci il primo valore: ");
47         scanf("%d",&puntatoreInizio->valore);
48         puntatore = puntatoreInizio;
49     }
50
51     for (i=2;i<=numElementi;i++){
52         puntatore->next = (struct Elemento *)malloc(sizeof(struct Elemento));
53         if (puntatore->next == NULL){
54             printf("\nNon è possibile allocare altra memoria...");
55             return puntatoreInizio;
56         }
57         puntatore = puntatore->next;
58         printf("\nInserisci elemento %d:",i);
59         scanf("%d",&puntatore->valore);
60     }
61     puntatore->next = NULL;
62     return puntatoreInizio;
63 }

```

Funzione che crea una lista e ritorna un puntatore al primo elemento

Liste.....esempio

Creazione e stampa di una semplice lista

```
23 void stampaLista(struct Elemento *puntatore){  
24     printf("\n\nListaElementi-->");  
25     while(puntatore != NULL){  
26         printf("%d-->",puntatore->valore);  
27         puntatore = puntatore->next;  
28     }  
29     printf("NULL\n");  
30 }
```

Esercizio 4

- Si scriva un programma che permetta all'utente di inserire delle Automobili (l'utente ne può inserire quante ne preferisce)
- Ogni auto deve contenere i campi *prezzo* e *modello*
- Alla fine il programma deve stampare tutte le auto inserite dall'utente

Esercizio 4

Soluzione (1/3)

```
3 struct Automobile{
4     char modello[20];
5     float prezzo;
6     struct Automobile *next;
7 };
8
9 struct Automobile *creaLista();
10 void stampaLista(struct Automobile *p);
11
12 int main(){
13     printf("Inizio programma....\n");
14     struct Automobile *p;
15     p = creaLista();
16     stampaLista(p);
17
18     return 0;
19 }
```

Dichiarazione della **struct** Automobile
(rappresenta un singolo elemento della lista)

Prototipi delle funzioni per
creare e leggere una lista

Assegno al puntatore *p* la lista
creata e ritornata dalla funzione

Esercizio 4 Soluzione (2/3)

```
21 struct Automobile *creaLista(){
22     struct Automobile *inizioLista =
23     (struct Automobile *)malloc(sizeof(struct Automobile));
24     if(inizioLista == NULL){
25         printf("Non è possibile creare la lista...\n");
26         return NULL;
27     }
28     printf("Auto 1:\nModello: ");
29     scanf("%s",&inizioLista->modello);
30     printf("\nPrezzo: ");
31     scanf("%f",&inizioLista->prezzo);
32     inizioLista->next = NULL;
33
34     struct Automobile *ptr;
35     ptr = inizioLista;
36
37     int scelta = 0, numAuto = 1;
38     while(1){
39         printf("\nDigita 0 per inserire un'altra auto: ");
40         scanf("%d",&scelta);
41         if(scelta!=0)break;
42
43         ptr->next = (struct Automobile *)malloc(sizeof(struct Automobile));
44         if(ptr->next == NULL){
45             printf("\nNon è possibile allocare altro spazio...\n");
46             return NULL;
47         }
48
49         printf("\nAuto %d:\nModello: ",++numAuto);
50         scanf("%s",&ptr->next->modello);
51         printf("\nPrezzo: ");
52         scanf("%f",&ptr->next->prezzo);
53         ptr = ptr->next;
54         ptr->next = NULL;
55
56     }
57
58     return inizioLista;
```

Creazione del primo elemento della lista

Creo un nuovo puntatore per scorrere la lista ed inserire nuovi elementi.
*N.B. il puntatore **inizioLista** non viene mai modificato per non perdere il riferimento all'inizio della lista!*

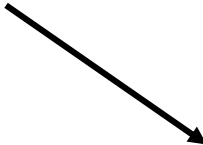
Il campo *next* dell'ultimo elemento deve essere impostato sempre a *NULL* per definire la fine della lista...

Ritorno il puntatore al primo elemento della lista

Esercizio 4

Soluzione (3/3)

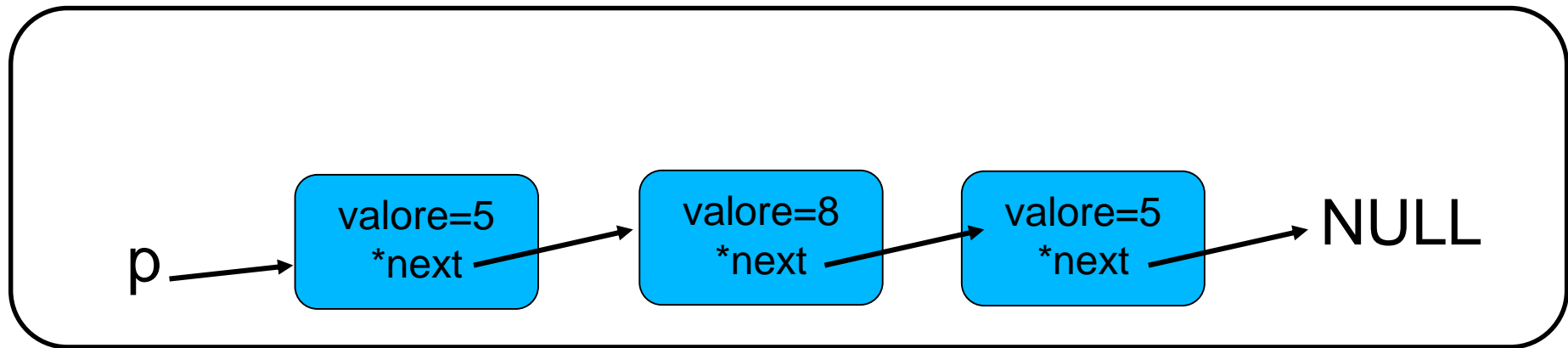
```
63 void stampaLista(struct Automobile *p){
64     printf("\nStampa delle auto...\n");
65     if (p == NULL){
66         printf("Lista vuota...\n");
67         return;
68     }
69     int numAuto = 1;
70     while(p!=NULL){
71         printf("\nAuto %d:\nModello: %s, Prezzo: %.2f",
72             numAuto++, p->modello, p->prezzo);
73         p = p->next;
74     }
75 }
76 }
```



Scorro la lista fino a che $p \neq \text{NULL}$

Liste Eliminare un elemento

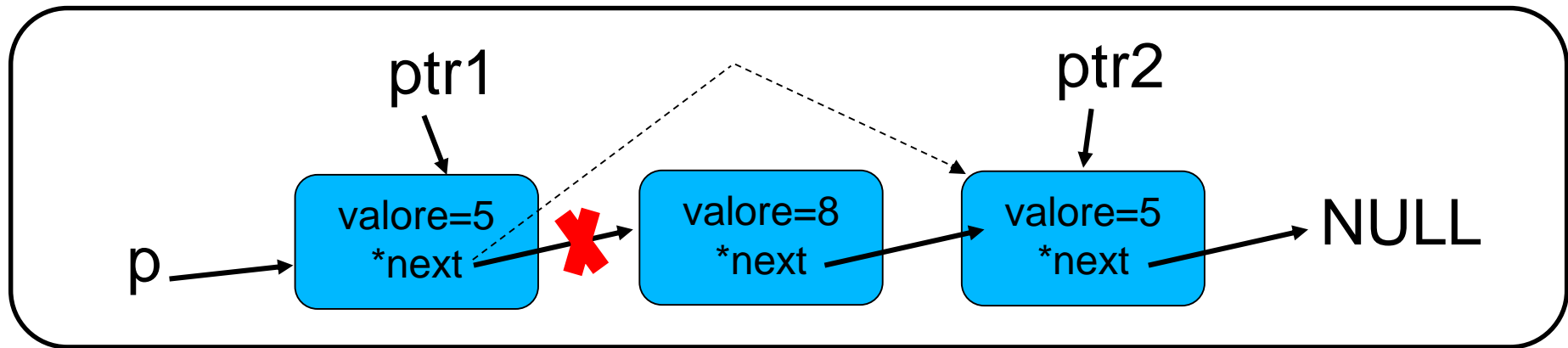
- Sia data una lista di tre **struct** contenenti ognuna, un campo intero *valore* e un *puntatore* al prossimo elemento.
- Il puntatore p punta al primo elemento della lista
- Come eliminare il secondo elemento?



Liste Eliminare un elemento

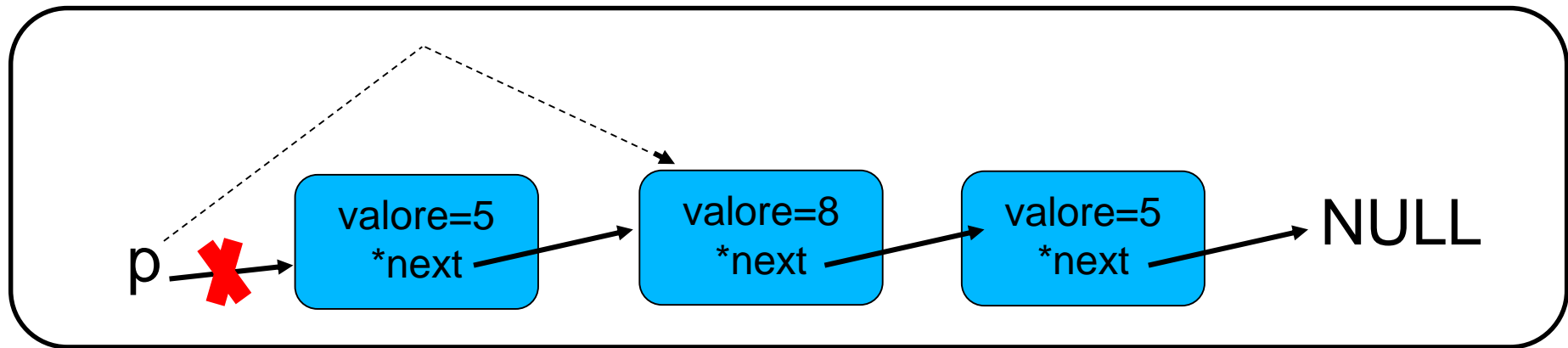
- 1- Dichiarare un puntatore (*ptr*) e farlo puntare all'elemento precedente a quello che si vuole eliminare
- 3- Usare l'istruzione:

ptr1 -> next = ptr1 -> next -> next;



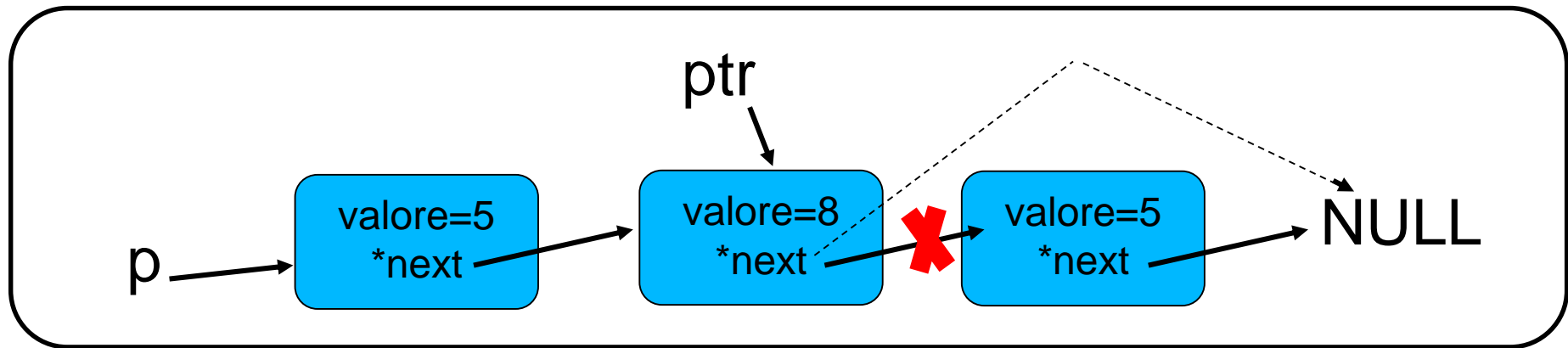
Liste Eliminare un elemento all'inizio

- Si vuole eliminare il primo elemento della lista
- Modificare il puntatore al primo elemento con la seguente istruzione:
- $p = p \rightarrow next;$



Liste Eliminare un elemento alla fine della lista

- Si vuole eliminare l'ultimo elemento della lista
- Dichiarare un secondo puntatore *ptr* che punta al primo elemento
- Scorrere la lista con *ptr* fino al penultimo elemento
- Usare la seguente istruzione:
ptr -> next = NULL;



Esercizio 5

- Si scriva un programma che crea una lista di 20 elementi, ognuno composto da un solo campo intero di nome *valore*
- Ogni elemento della lista deve impostare il valore alla rispettiva posizione nella lista
- *Es. il primo elemento deve avere valore 1, il secondo deve avere valore 2 ecc...*
- Si dichiarino le seguenti funzioni:

```
struct Elemento *creaLista();
```

```
struct Elemento *eliminaPrimo(struct Elemento *p);
```

```
void eliminaDecimo(struct Elemento *p);
```

```
void eliminaUltimo(struct Elemento *p);
```

```
void stampaLista(struct Elemento *p);
```

- Il programma, una volta creata la lista con la relativa funzione, usa le funzioni dichiarate rispettivamente per eliminare il primo, il decimo e l'ultimo elemento. Ogni volta che elimina un elemento (richiamando la funzione corretta), deve richiamare la funzione *stampaLista* per stampare l'intera lista modificata.

Esercizio 5

Soluzione(1/3)

```
3 typedef struct Elemento{
4     int valore;
5     struct Elemento *next;
6 };
7
```

Creazione della **struct** (elemento della lista)

```
8 struct Elemento *creaLista();
9 struct Elemento *eliminaPrimo(struct Elemento *p);
10 void eliminaDecimo(struct Elemento *p);
11 void eliminaUltimo(struct Elemento *p);
12 void stampaLista(struct Elemento *p);
13
```

Prototipi delle funzioni

```
14 int main(){
15
16     struct Elemento *p = creaLista();
17     printf("Stampo la lista originale: \n");
18     stampaLista(p);
19     p = eliminaPrimo(p);
20     printf("\nStampo la lista senza il primo elemento: \n");
21     stampaLista(p);
22     eliminaDecimo(p);
23     printf("\nStampo la lista senza il primo e il decimo elemento: \n");
24     stampaLista(p);
25     eliminaUltimo(p);
26     printf("\nStampo la lista senza il primo, il decimo e l'ultimo elemento: \n");
27     stampaLista(p);
28 }
```

Chiamate alle rispettive funzioni...

Esercizio 5 Soluzione(2/3)

```
32 struct Elemento *creaLista(){
33     struct Elemento *ptr, *ptr2;
34     ptr = (struct Elemento *)malloc(sizeof(struct Elemento));
35     if(ptr==NULL){
36         printf("Non è possibile creare la lista...\n");
37         return NULL;
38     }
39     ptr->valore = 1;
40     ptr2 = ptr;
41     int i = 0;
42     for (i=1;i<20;i++){
43         ptr2->next = (struct Elemento *)malloc(sizeof(struct Elemento));
44         if(ptr2->next == NULL){
45             printf("Non è possibile aggiungere elementi alla lista...\n");
46             return NULL;
47         }
48         ptr2 = ptr2->next;
49         ptr2->valore = i + 1;
50         ptr2->next = NULL;
51     }
52     return ptr;
53 }
54
55
56
57 struct Elemento *eliminaPrimo(struct Elemento *p){
58     if(p==NULL){
59         printf("\neliminaPrimo--> Lista vuota...\n");
60         return NULL;
61     }
62     p = p->next;
63     return p;
64 }
```

Crea la lista e
ritorna il puntatore
al primo elemento

Elimina il primo
elemento e ritorna il
puntatore al "nuovo"
primo elemento

Esercizio 5 Soluzione(3/3)

```
66 void eliminaDecimo(struct Elemento *p){
67     if(p==NULL){
68         printf("\neliminaDecimo--> Lista vuota...\n");
69         return NULL;
70     }
71
72     while(p->next->valore != 10){
73         p=p->next;
74     }
75
76     p->next = p->next->next;
77
78 }
79
80 void eliminaUltimo(struct Elemento *p){
81     if(p==NULL){
82         printf("\neliminaUltimo--> Lista vuota...\n");
83         return NULL;
84     }
85
86     while(p->next->next != NULL){
87         p=p->next;
88     }
89
90     p->next = NULL;
91 }
92
93 void stampaLista(struct Elemento *p){
94     while (p!=NULL){
95         printf("-->%d",p->valore);
96         p = p->next;
97     }
98 }
99
100
```

Elimina il decimo
elemento della lista

Elimina l'ultimo
elemento della lista

Stampa la lista
puntata dal
parametro *p

Esercizio 6

- Scrivere un programma che gestisce una lista composta da Elementi con un solo campo intero di nome *valore*
- La lista deve essere gestita con una politica *LIFO* (*Last In First Out*), cioè l'ultimo elemento inserito è il primo che può essere prelevato
- Implementare le seguenti funzioni:

```
struct Elemento *push(struct Elemento *p);  
struct Elemento *pop(struct Elemento *p);  
void stampaLista(struct Elemento *p);
```

- *pop*: toglie dalla lista l'ultimo elemento inserito
- *push*: inserisce un nuovo elemento nella lista
- *stampaLista*: stampa tutti gli elementi della lista
- Nella funzione main, fornire un semplice menù che permette all'utente di scegliere se fare una *pop* o una *push* e dopo ogni modifica della lista, chiama automaticamente la funzione per stampare a video la lista.

Esercizio 6

Soluzione(1/3)

```
3 struct Elemento{
4     int valore;
5     struct Elemento *next;
6 };
7
```

Dichiarazione
della struct

```
8 struct Elemento *push(struct Elemento *p);
9 struct Elemento *pop(struct Elemento *p);
10 void stampaLista(struct Elemento *p);
11
```

Prototipi della
funzioni

```
12 int main(){
13     printf("Inizio Programma....\n");
14     int scelta = 0;
15     struct Elemento *p = NULL;
16
17     while(1){
18         printf("\nMenu-->\n0)push\n1)pop\n2)Esci\n\nScelta: ");
19         scanf("%d",&scelta);
20         if(scelta == 2){
21             printf("\n\n....Chiusura programma....");
22             break;
23         }else if(scelta == 0){
24             p = push(p);
25         }else if (scelta == 1){
26             p = pop(p);
27         }
28         printf("\nLista modificata:");
29         stampaLista(p);
30     }
31
32     return 0;
33 }
```

Menù che chiede
all'utente se fare
una *pop* o una *push*

Esercizio 6 Soluzione(2/3)

```
35 struct Elemento *push(struct Elemento *p){
36     if(p == NULL){
37         p = (struct Elemento *)malloc(sizeof(struct Elemento));
38         if(p == NULL){
39             printf("\nImpossibile aggiungere elemento...\n");
40             return NULL;
41         }
42         printf("\nInserisci il valore: ");
43         scanf("%d",&p->valore);
44         p->next = NULL;
45         return p;
46     }
47     struct Elemento *ptr = p;
48     while(ptr->next != NULL){
49         ptr = ptr->next;
50     }
51     ptr->next = (struct Elemento *)malloc(sizeof(struct Elemento));
52     if(ptr->next == NULL){
53         printf("\nImpossibile aggiungere elemento...\n");
54         return p;
55     }
56     printf("\nInserisci il valore: ");
57     scanf("%d",&ptr->next->valore);
58     ptr->next->next = NULL;
59
60     return p;
61 }
```

La lista è vuota.
Dobbiamo inserire il
primo elemento

Esercizio 6

Soluzione(3/3)

```
63 struct Elemento *pop(struct Elemento *p){
64     if(p == NULL){
65         printf("\nLista vuota! ");
66         return NULL;
67     }
68     if (p->next == NULL){
69         free(p);
70         return NULL;
71     }
72
73     struct Elemento *ptr1 = p, *ptr2 = p->next;
74
75     while(ptr2->next != NULL){
76         ptr1 = ptr1->next;
77         ptr2 = ptr2->next;
78     }
79     ptr1->next = NULL;
80     free(ptr2);
81
82     return p;
83 }
84
85 void stampaLista(struct Elemento *p){
86     printf("\n-->");
87     while(p!=NULL){
88         printf("%d-->",p->valore);
89         p = p->next;
90     }
91     printf("NULL\n\n");
92
93 }
```

Lista vuota!

È presente un solo elemento

Esercizio 7

- Scrivere un programma che gestisce una coda composta da Elementi con un solo campo intero di nome *valore*
- La lista deve essere gestita con una politica *FIFO (First In First Out)*, cioè il primo elemento inserito è il primo che può essere prelevato
- Implementare le seguenti funzioni:

```
struct Elemento *push(struct Elemento *p);  
struct Elemento *pop(struct Elemento *p);  
void stampaLista(struct Elemento *p);
```

- *pop*: toglie dalla lista il primo elemento inserito
- *push*: inserisce un nuovo elemento nella lista
- *stampaLista*: stampa tutti gli elementi della lista
- Nella funzione main, fornire un semplice menù che permette all'utente di scegliere se fare una *pop* o una *push* e ad ogni modifica della lista la stampa a video.

Esercizio 7 Soluzione (simile all'esercizio precedente)

```
35 struct Elemento *push(struct Elemento *p){
36     if(p == NULL){
37         p = (struct Elemento *)malloc(sizeof(struct Elemento));
38         if(p == NULL){
39             printf("\nImpossibile aggiungere elemento...\n");
40             return NULL;
41         }
42         printf("\nInserisci il valore: ");
43         scanf("%d",&p->valore);
44         p->next = NULL;
45         return p;
46     }
47     struct Elemento *ptr = (struct Elemento *)malloc(sizeof(struct Elemento));
48     if(ptr == NULL){
49         printf("\nImpossibile aggiungere elemento...\n");
50         return p;
51     }
52     ptr->next = p;
53     printf("\nInserisci il valore: ");
54     scanf("%d",&ptr->valore);
55
56     return ptr;
57 }
```

Nell'esercizio precedente inseriamo e preleviamo gli elementi in coda (alla fine della lista).

Adesso dobbiamo prelevare in coda ed inserire in testa...

Quindi la soluzione è uguale ma dobbiamo modificare la funzione *push* affinché aggiunga il nuovo elemento in testa

Creiamo un nuovo elemento (puntato da *ptr*) e facciamo puntare il suo campo *next* al primo elemento della lista. Alla fine ritorniamo lo stesso puntatore *ptr* (che punta al nuovo elemento, che sarà quindi il primo)