Esercitazione di Reti degli elaboratori

Prof.ssa Chiara Petrioli

SAPIENZA
UNIVERSITÀ DI ROMA

Laboratorio di C

Luca Iezzi
iezzi@diag.uniroma1.it

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

```c
struct Articolo {
    int codice;
    float prezzo;
};

int main(){
    //dichiarazione e inizializzazione
    struct Articolo art1 = { 467, 12.5};

    //dichiarazione
    struct Articolo art2;
    //accesso ai campi in scrittura
    art2.codice = 189;
    art2.prezzo = 34.99;

    //accesso ai campi in lettura
    printf("art1 - codice: %d, prezzo: %0.2f \n",art1.codice, art1.prezzo);
    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
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);
```

art (Articolo)

- codice=1234
- prezzo=12.99

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 dichiari 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

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)*

Esempio

- Si scriva un programma che dichiara 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

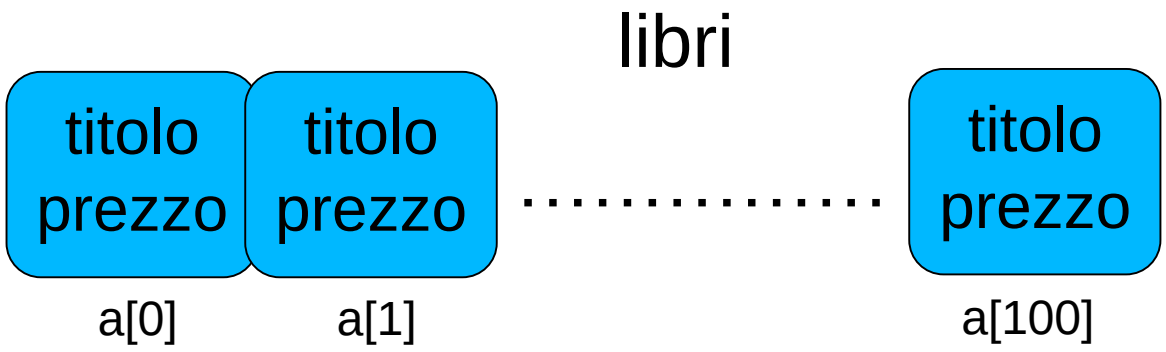
Esempio Soluzione

```c
typedef struct Libro{
    char titolo[50];
    float prezzo;
}Libro;
int controlloLibri(Libro l1, Libro l2);
int main(){
    Libro libro1,libro2;
    printf("Libro1-->");
    printf("\ntitolo: ");
    scanf("%s",&libro1.titolo);
    printf("\nprezzo: ");
    scanf("%f",&libro1.prezzo);
    printf("Libro2-->");
    printf("\ntitolo: ");
    scanf("%s",&libro2.titolo);
    printf("\nprezzo: ");
    scanf("%f",&libro2.prezzo);
    if(controlloLibri(libro1,libro2)==0) printf("I libri sono uguali!");
    else printf("I libri sono diversi!");
    return 0;
}
int controlloLibri(Libro l1, Libro l2){
    if ( (l1.prezzo==l2.prezzo) && (strcmp(l1.titolo,l2.titolo)==0) ){
        return 0;
    }
    return 1;
}
```

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

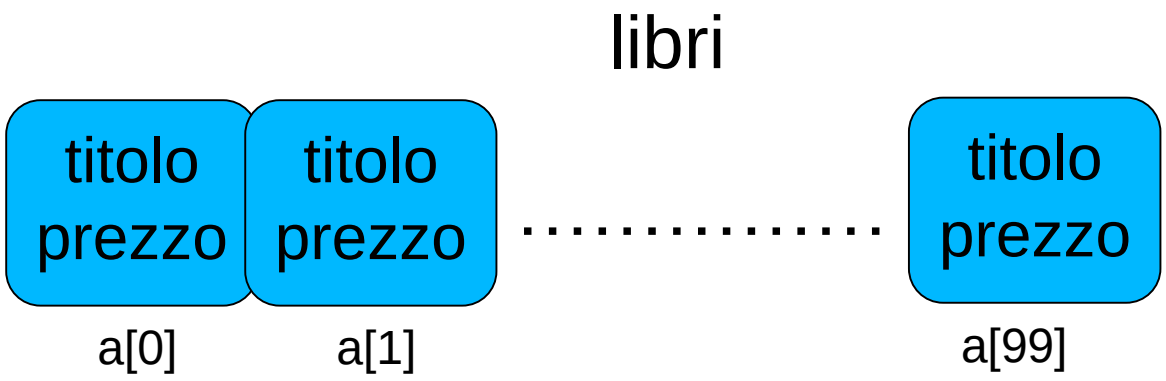


- L'accesso è lo stesso come con gli array di tipi primitivi

Array di Struct

Accedere ai campi

- Accedere ai campi della struttura alla posizione *i* , *tale che* *0 <= 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

```c
union Dato {
    int i;
    float f;
    char str[20];
};

int main( ) {

    union Dato dato;

    dato.i = 34;
    dato.f = 12.37;
    strcpy( dato.str, "Programmazione C");

    printf( "dato.i : %d\n", dato.i);
    printf( "dato.f : %f\n", dato.f);
    printf( "dato.str : %s\n", dato.str);
```

Che cosa stampano le tre funzioni *printf* ?

Union

Un altro esempio

```c
union Dato {
    int i;
    float f;
    char str[20];
};

int main( ) {

    union Dato dato;

    dato.i = 34;
    dato.f = 12.37;
    strcpy( dato.str, "Programmazione C");

    printf( "dato.i : %d\n", dato.i);
    printf( "dato.f : %f\n", dato.f);
    printf( "dato.str : %s\n", dato.str);
```

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

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

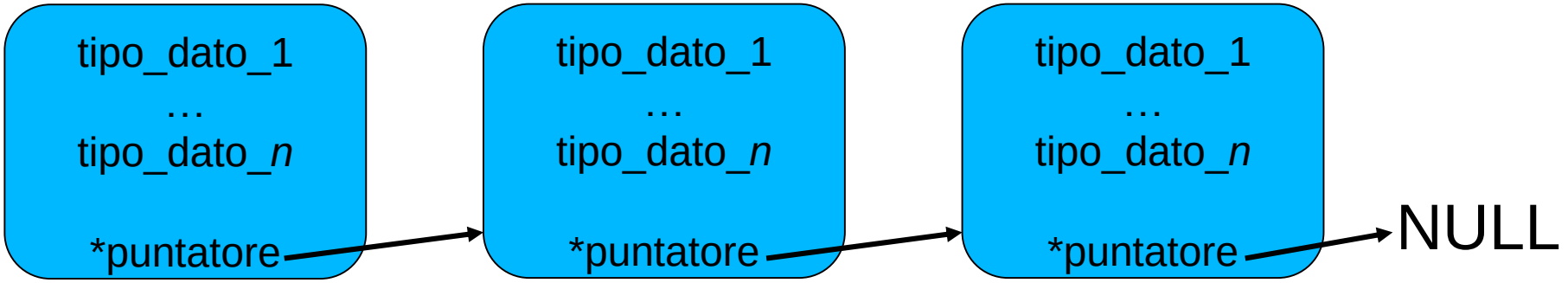
```
dato.i : 1735357008
dato.f : 1130754282837771100000000.000000
dato.str : Programmazione C
```

Esercizio 3

- Si scriva un programma che dichiara 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

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!**

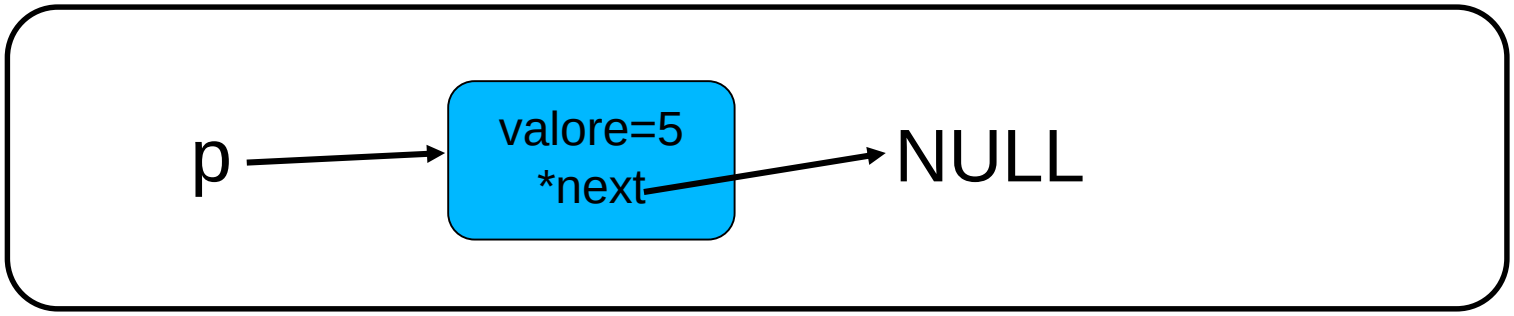
Liste

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…

**next* punta al prossimo *elemento*

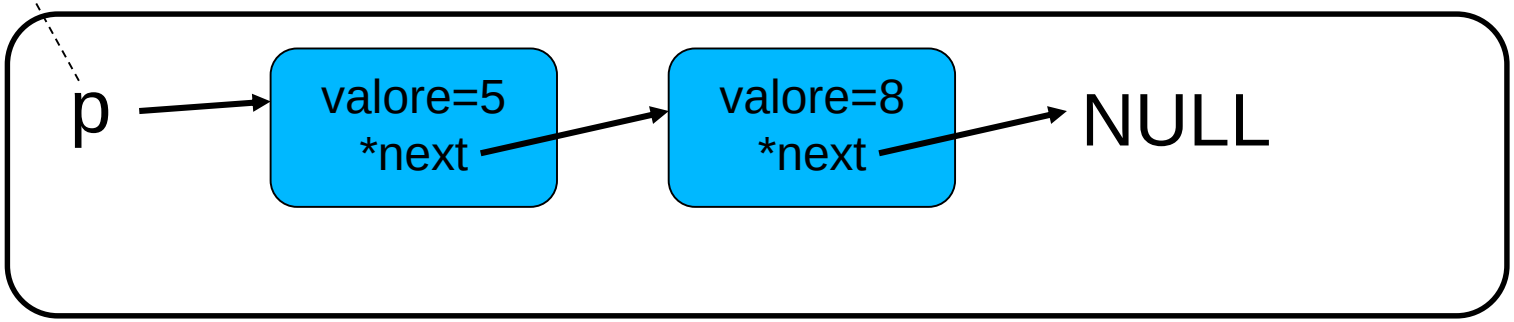
Allocato spazio per una struttura di tipo Elemento

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

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

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

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

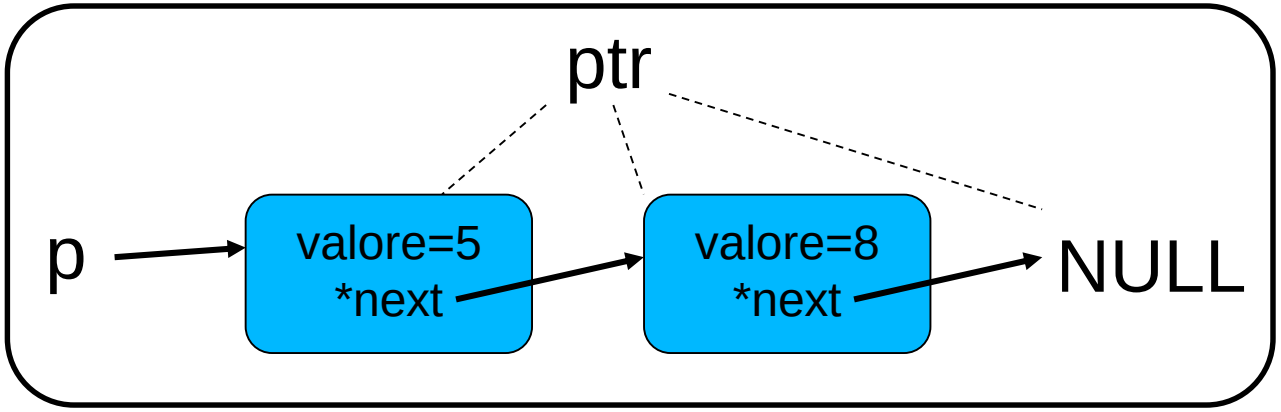


Liste Scorrimento…

```
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

```c
struct Elemento *creaLista(){

    struct Elemento *puntatoreInizio, *puntatore;
    int i, numElementi;
    printf("Inserire numero elementi: ");
    scanf("%d",&numElementi);

    if(numElementi==0) puntatoreInizio == NULL;
    else {
        puntatoreInizio = (struct Elemento*) malloc(sizeof(struct Elemento));
        if(puntatoreInizio==NULL){
            printf("\nNon è possibile allocare spazio! Lista vuota...");
            return puntatoreInizio;
        }
        printf("Inserisci il primo valore: ");
        scanf("%d",&puntatoreInizio->valore);
        puntatore = puntatoreInizio;
    }

    for (i=2;i<=numElementi;i++){
        puntatore->next = (struct Elemento *)malloc(sizeof(struct Elemento));
        if (puntatore->next == NULL){
            printf("\nNon è possibile allocare altra memoria...");
            return puntatoreInizio;
        }
        puntatore = puntatore->next;
        printf("\nInserisci elemento %d:",i);
        scanf("%d",&puntatore->valore);
    }
    puntatore->next = NULL;
    return puntatoreInizio;
}
```

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

Liste……esempio

Creazione e stampa di una semplice lista

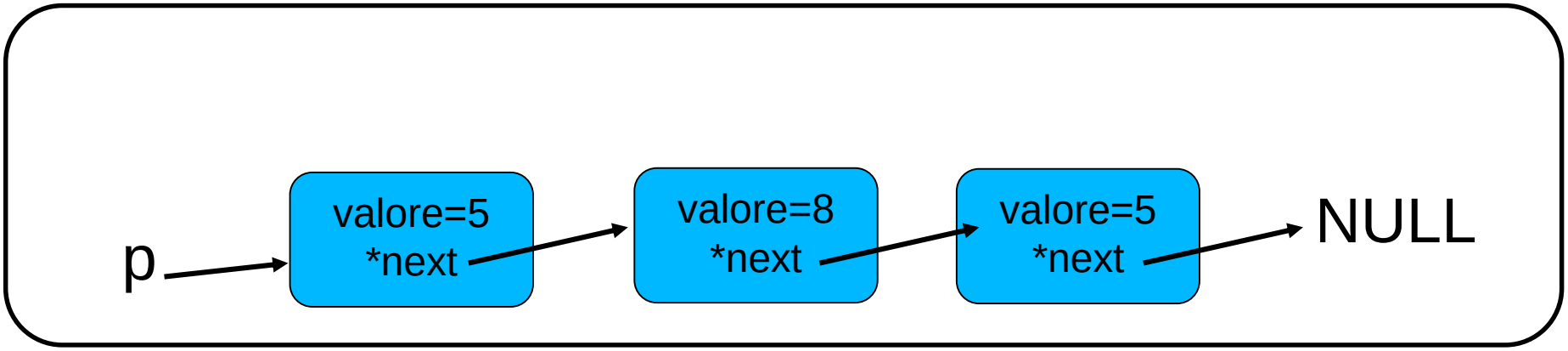
```
23  void stampaLista(struct Elemento *puntatore){
24      printf("\n\nListaElementi-->");
25      while(puntatore != NULL){
26          printf("%d-->",puntatore->valore);
27          puntatore = puntatore->next;
28      }
29      printf("NULL\n");
30  }
```

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

Eliminare un elemento

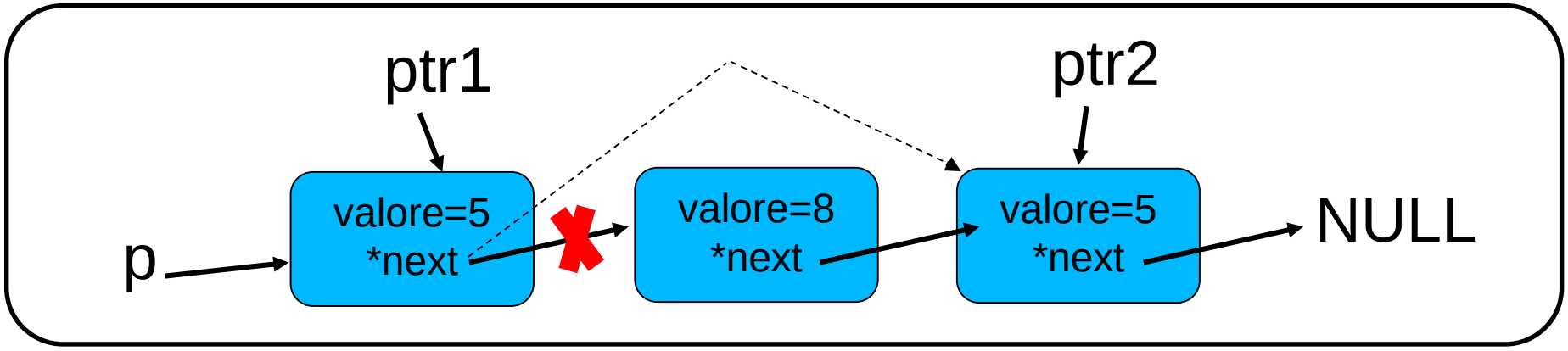
- Sia data una lista di tre **struct** contenenti ognuna, un campo intero *valore* e un *puntatore* al prossimo elemento.
- Il puntatore p punta al primo elemento della lista
- Come eliminare il secondo elemento?



Eliminare un elemento

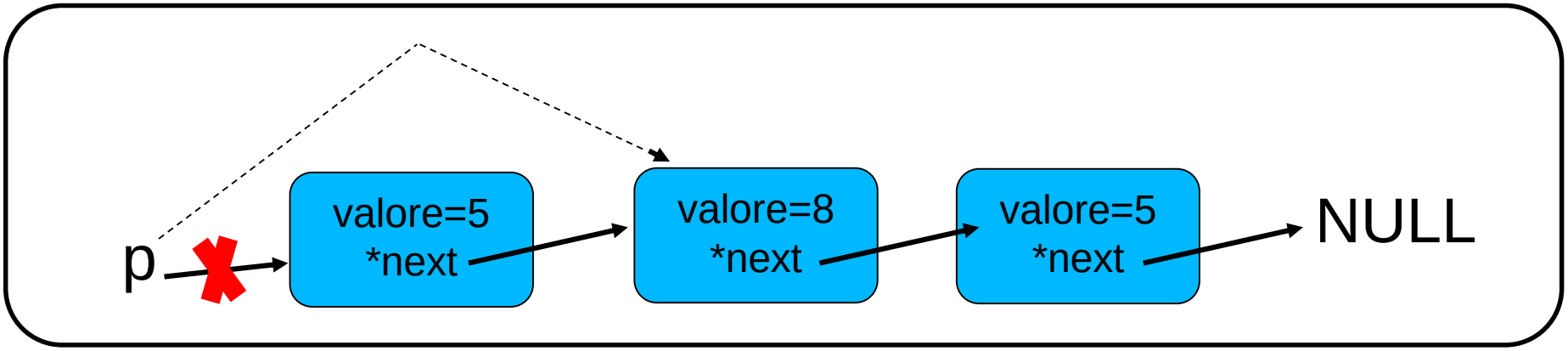
- 1- Dichiarare un puntatore (*ptr*) e farlo puntare all'elemento precedente a quello che si vuole eliminare
- 3- Usare l'istruzione:

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



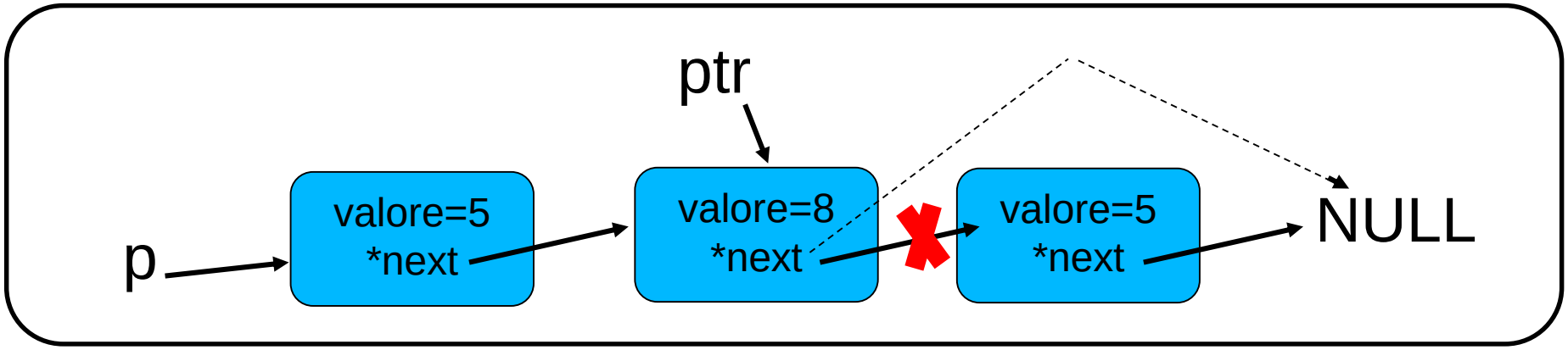
Eliminare un elemento all'inizio

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



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 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 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.