UNIVERSITÀ PARTHENOPE
DiST
Dipartimento di Scienze e Tecnologie

Programmazione II e Laboratorio di PII

Strutture Dati elementari

Angelo Ciaramella

Strutture dati dinamiche

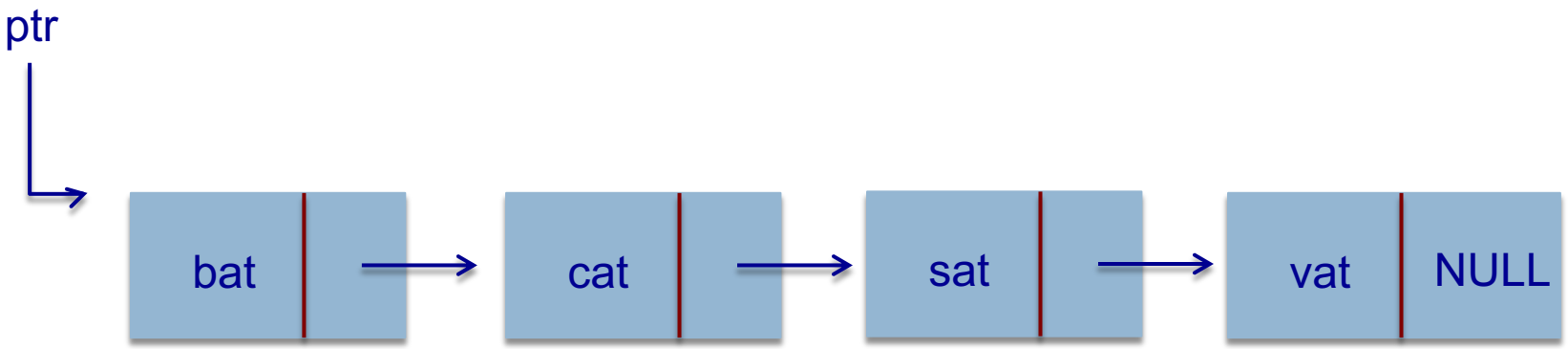
- Le rappresentazioni sequenziali dei dati (i.e., array) sono adeguate per molte operazioni

- Tuttavia, usando una rappresentazione sequenziale alcune operazioni diventano poco efficienti
  - Inserimento o cancellazione in una lista ordinata

Liste concatenate

- Una lista concatenata è una collezione di dati con organizzazione lineare
- Essa è formata da una sequenza ordinata di nodi con i collegamenti (link) rappresentati da frecce
- Il nome del puntatore al primo nodo è il nome della lista

Liste concatenate



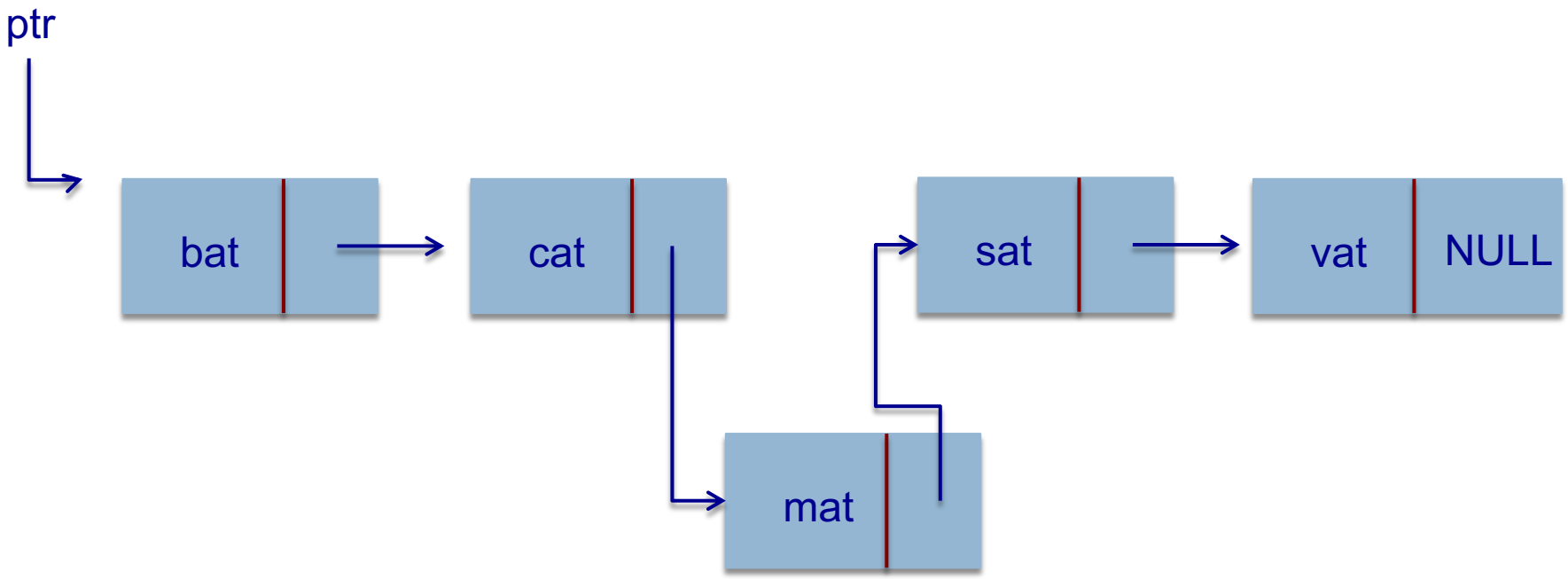
Tipico modo di rappresentare una lista singolarmente concatenata

Caratteristiche

- Una lista concatenata ha le seguenti proprietà
  - I nodi non risiedono in locazioni sequenziali
  - Le locazioni dei nodi possono cambiare da un'esecuzione all'altra

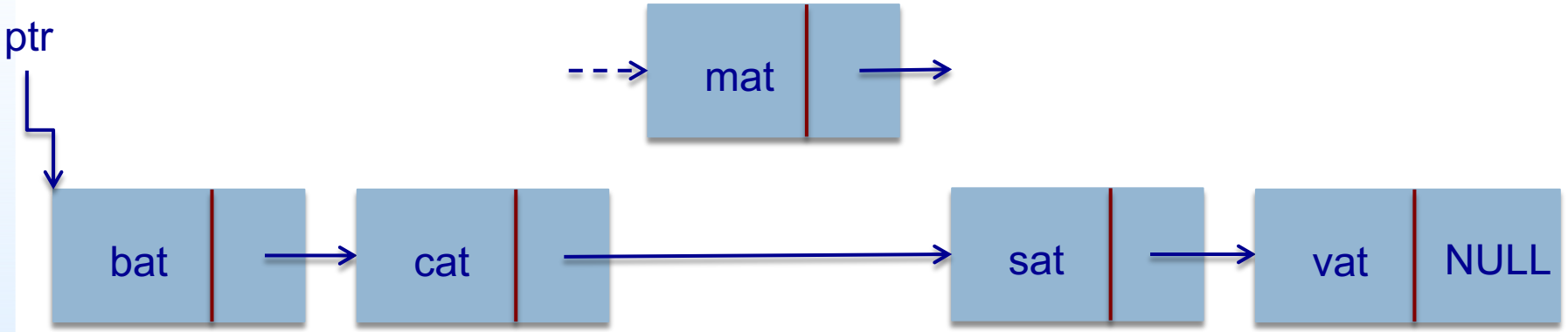
- E' più semplice effettuare cancellazioni e inserimenti arbitrari rispetto ad una lista sequenziale

Inserimento di un elemento



Inserimento di *mat* dopo *cat*

Cancellazione di un elemento



Cancellazione di *mat* dalla lista

Rappresentazione della lista

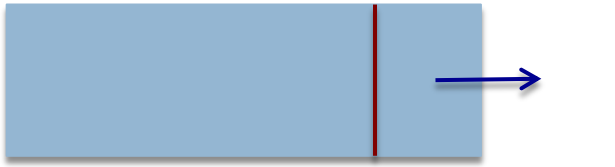
- Da questa breve analisi possiamo osservare che occorrono I seguenti elementi
  - Un meccanismo per definire la struttura di un nodo, cioè i campi che esso contiene
  - Una tecnica per creare nuovi nodi
  - Una tecnica per eliminare nodi

Struttura dei nodi

- I nodi sono descritti da strutture autoreferenziali

- Una struttura autoreferenziale contiene un membro puntatore che punta a una struttura dello stesso tipo

Rappresentazione grafica



Struttura di tipo *nodo*

Struttura dei nodi in C

```
struct nodo {
          int dato;
          struct nodo *next_nodo;
 };
```


Strutture autoreferenziali

- In base a questa dichiarazione `struct nodo` ha due membri: dato e puntatore

- Il membro `next_nodo` punta a una struttura di tipo `struct nodo`
  - Struttura dello stesso tipo di quella dichiarata (struttura autoreferenziale)
  - `next_nodo` è chiamato link o collegamento
  - Tramite `next_nodo` ogni struttura viene collegata alla struttura successiva

- Le strutture autoreferenziali possono essere collegate insieme per formare strutture dati come
  - liste, code, pile e alberi

Struttura dei nodi in C++

```
class Node{

public:
   Node(int); // costruttore
   void setData(int); // imposta il dato membro
   int getData() const; // legge il dato membro
   void setNextPtr(const Node *); // imposta il puntatore al
   prossimo nodo
   const Node *getNextPtr() const; // ottiente il puntatore
   al prossimo nodo

private:
   int data; //dato memorizzato in questo nodo
   Node *nextPtr; //puntatore ad un altro oggetto della
   stessa classe
}
```


Creare nodi

- Creare e mantenere strutture dinamiche di dati richiede l'allocazione dinamica di memoria
  - Ottenere un maggiore spazio di memoria al momento dell'esecuzione
  - Liberare lo spazio non più necessario

Creare nodi

- Il sistema fornisce `malloc()` nella libreria standard il cui prototipo si trova in stdlib.h
  - `malloc(size)`
    - Restituisce un puntatore a un'area di memoria di dimensione sufficiente per un oggetto di `size byte`
    - Tale area non viene inizializzata
    - Il parametro è di tipo `size_t` (intero senza segno definito in stddef.h e stdlib.h), il tipo del risultato dell'operatore `sizeof()`
    - Restituisce un puntatore a `void`, assegnabile a una variabile di qualunque tipo puntatore

Uso di malloc()

- Esempio di uso di `malloc()`

```
new_nodo = malloc(sizeof (struct nodo));
```

- Alloca nuova area nella memoria
- Memorizza un puntatore alla memoria allocata nella varible `new_nodo`
- `sizeof (struct nodo)` serve per determinare la dimensione in byte della struttura `struct nodo`
- Se non è disponibile alcuna memoria restituisce `NULL`

Liberare memoria

- La funzione free libera la memoria
  - La memoria viene restituita al sistema in modo che possa essere allocata in futuro

```
free(new_nodo)
```


Lista concatenata di interi

- Vogliamo creare una lista concatenata di numeri interi

- La struttura dei nodi è così definita

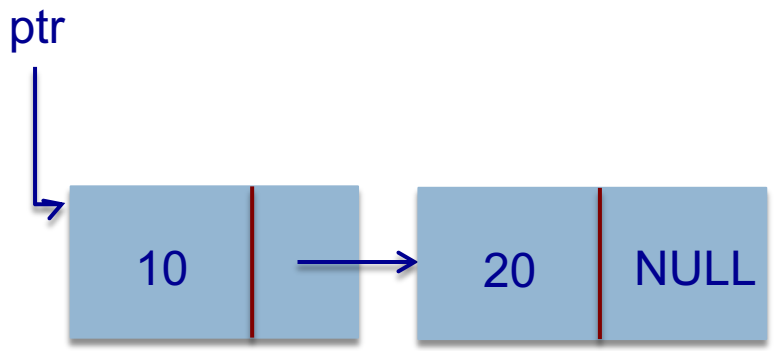
```
typedef struct list_node *list_pointer;
typedef struct list_node {
            int dati;
            list_pointer link;
            };
list_pointer ptr = NULL;
```


Lista concatenata con due nodi

```
list_pointer crea ()
{
/* crea una lista concatenata con due nodi */
list_pointer primo, secondo;

primo = (list_pointer) malloc (sizeof(list_node));
secondo = (list_pointer) malloc (sizeof(list_node));

secondo->link = NULL;
secondo->dati = 20;
primo->dati = 10;
primo->link = secondo;

return primo;
}
```

Funzione per la creazione di una lista a due nodi

Lista a due nodi



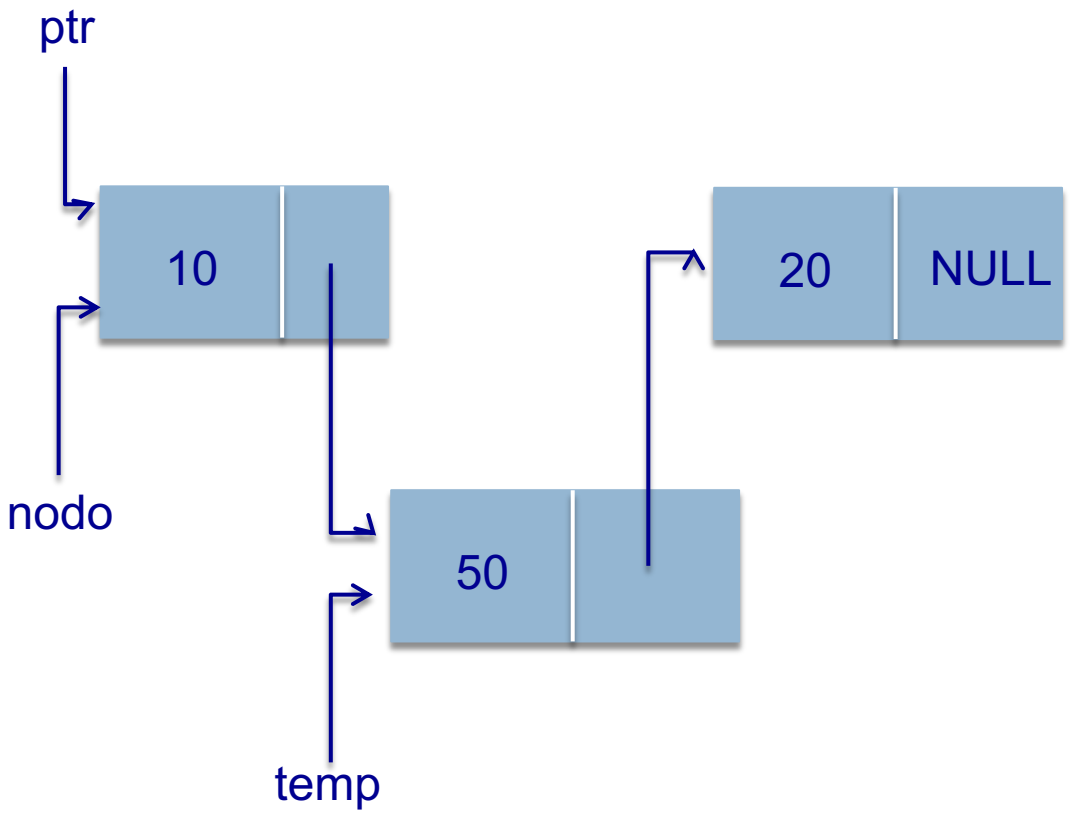
Creazione di due nodi

Esercizio: creare la stessa lista usando la classe in C++

Inserimento di un nodo

```
void inserisci (list_pointer *ptr, list_pointer nodo)
{
/* Inserisce un nuovo nodo con il campo dati = 50 nella lista ptr dopo
nodo */

list_pointer temp;

temp = (list_pointer) malloc (sizeof(list_node));

/* controllo IS_FULL(temp) */

if(*ptr) {
  temp->link = nodo->link;
  nodo->link = temp;
}
else {
  temp->link = NULL;
  *ptr = temp;
}
}
```


Lista a tre nodi



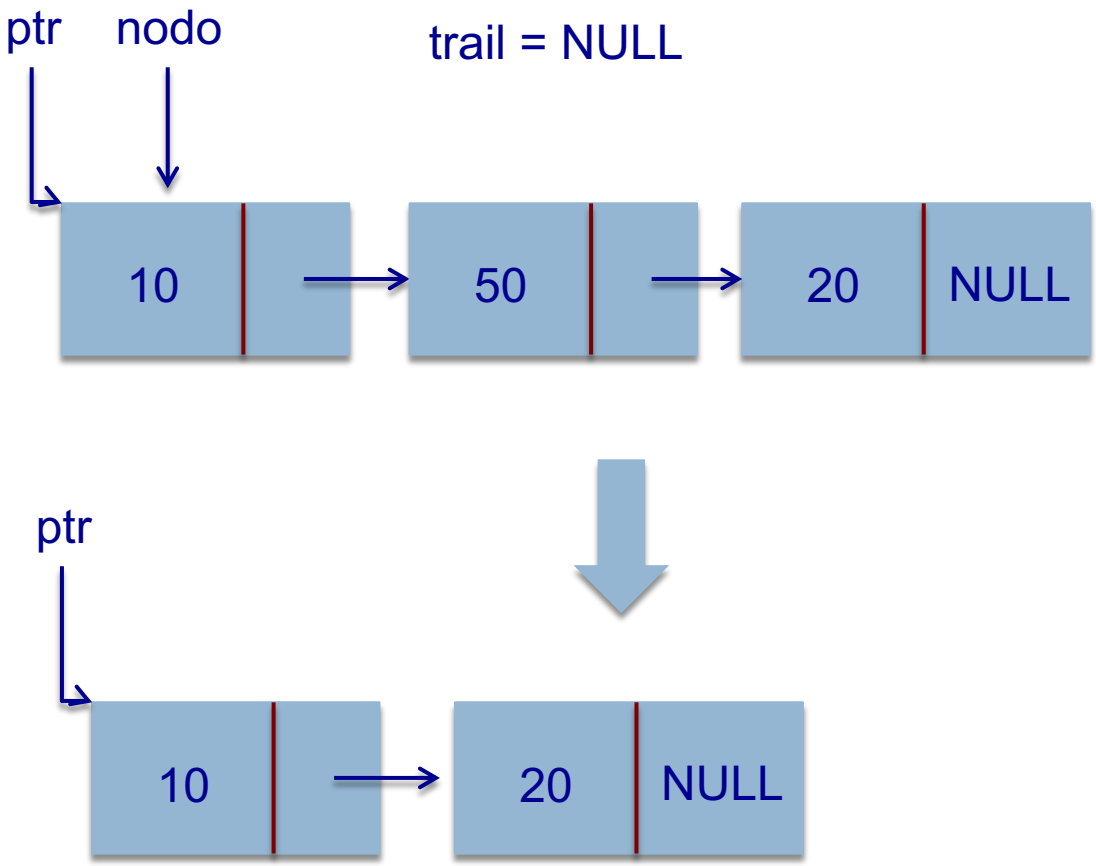
Inserimento di un nodo

Esercizio: creare la stessa lista usando la classe in C++

Cancellazione di un nodo

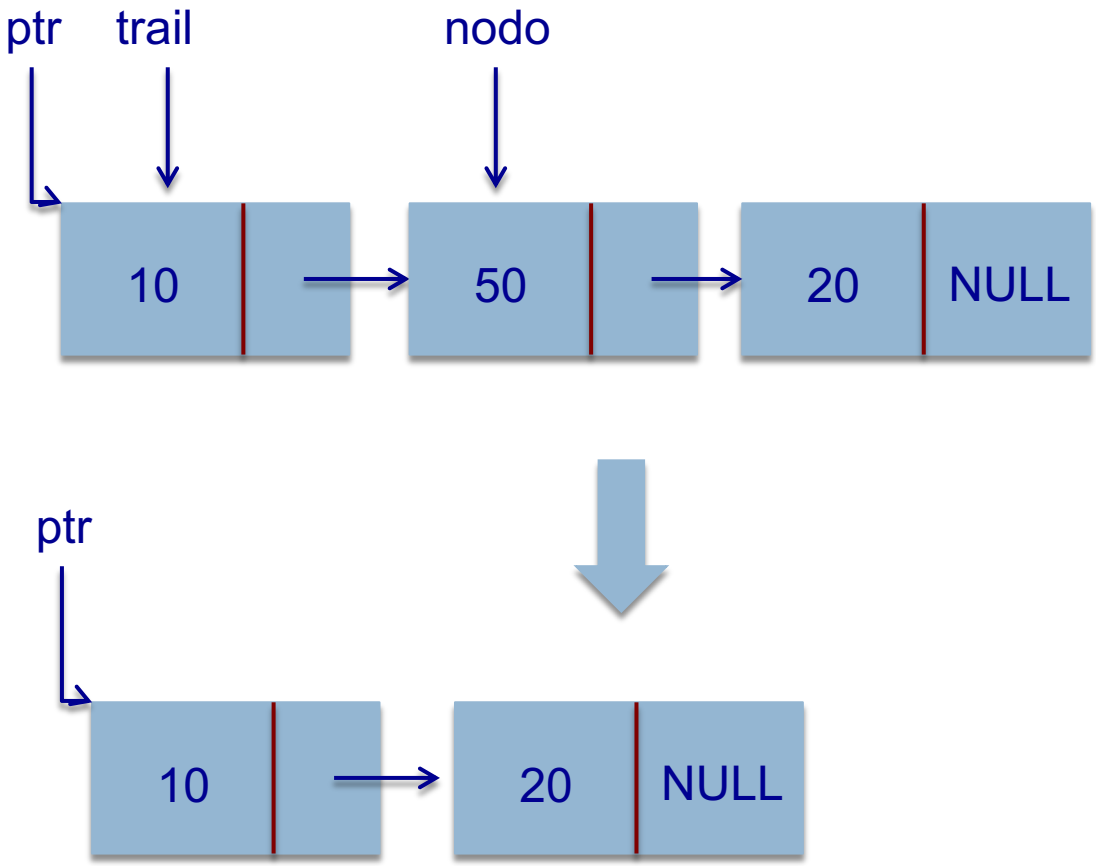
```
void delete (list_pointer *ptr, list_pointer trail,
list_pointer nodo)
{
/* Cancella un nodo dalla lista; trail e' il nodo
precedente; ptr e' l'inizio della lista */

if(trail)
  trail->link = nodo->link;
else
   *ptr = (*ptr)->link;

free(nodo);
}
```


Cancellazione di un nodo



Chiamata alla funzione *delete(&ptr, NULL, ptr)*

Cancellazione di un nodo



Chiamata alla funzione *delete(&ptr, ptr, ptr->link)*

Esercizio: creare la stessa lista usando la classe in C++