CORSO DI PROGRAMMAZIONE A-L

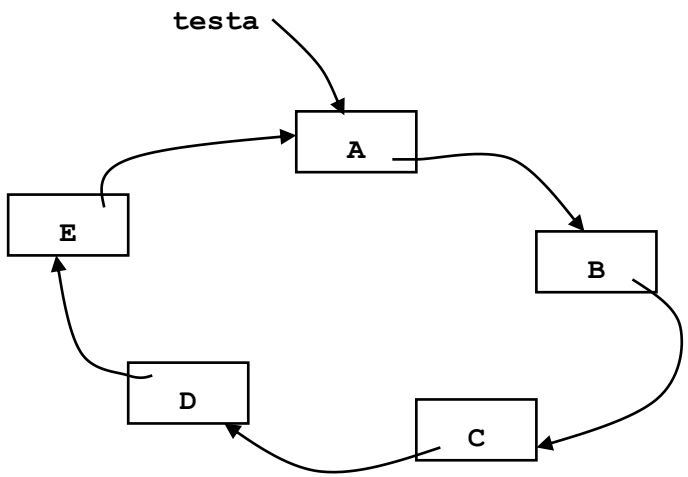
A.A. 2016-17

Dispensa 21

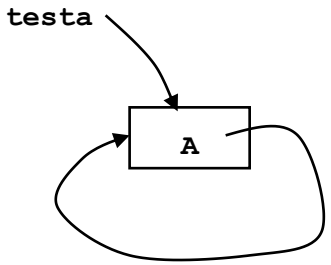
Dott. Mirko Ravaioli
e-mail: mirko.ravaioli@unibo.it

21 Liste Circolari

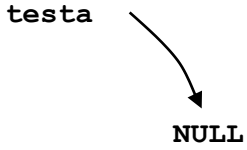
Una Lista Circolare è una lista semplice dove l'ultimo elemento è collegato con il primo:



Lista circolare con più elementi



Lista circolare con un solo elemento



Lista circolare vuota

La struttura dati per gestire ogni elemento (cella) della lista circolare (come per le liste):

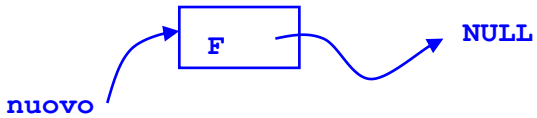
```
struct cella{
      char valore;
      struct cella *next;
};

struct cella *testa = NULL; //puntatore alla testa
```

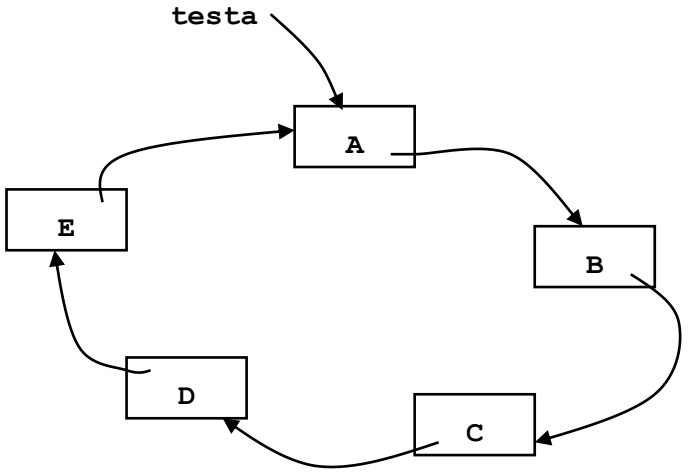
Tutti i nuovi elementi vengono inseriti in testa e tutte le altre operazioni vengono fatte sempre a partire dalla testa.

21.1 Inserimento di un nuovo elemento

Attraverso la funzione malloc() allochiamo lo spazio necessario per mantenere in memoria la nuova cella da inserire all'interno della lista:

1



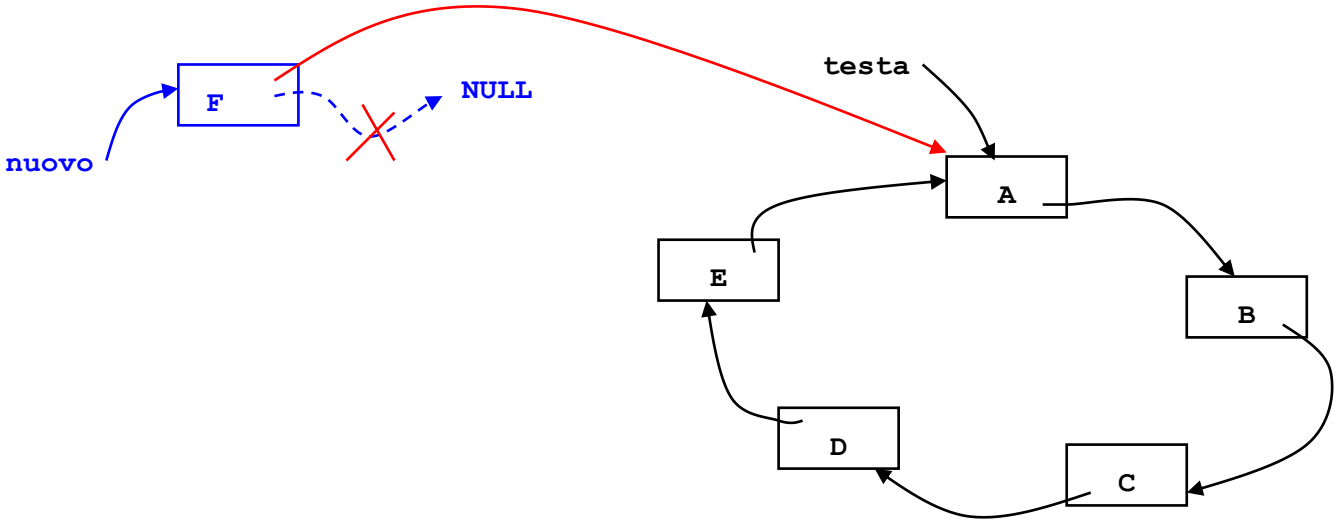



```
nuovo = (struct cella*)malloc(sizeof(struct cella));
```

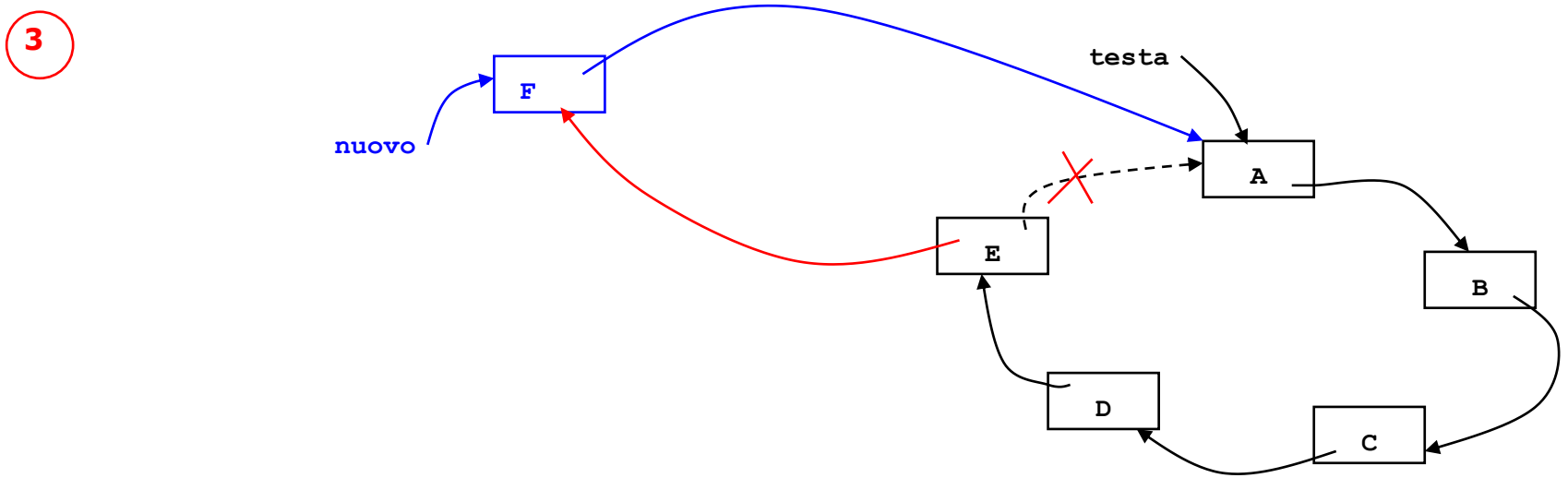
Dove "nuovo" è un puntatore di tipo struct cella

La seconda operazione da compiere è collegare la nuova cella alla prima della lista, quindi fare in modo che l'elemento "next" della nuova cella prenda lo stesso valore di "testa":

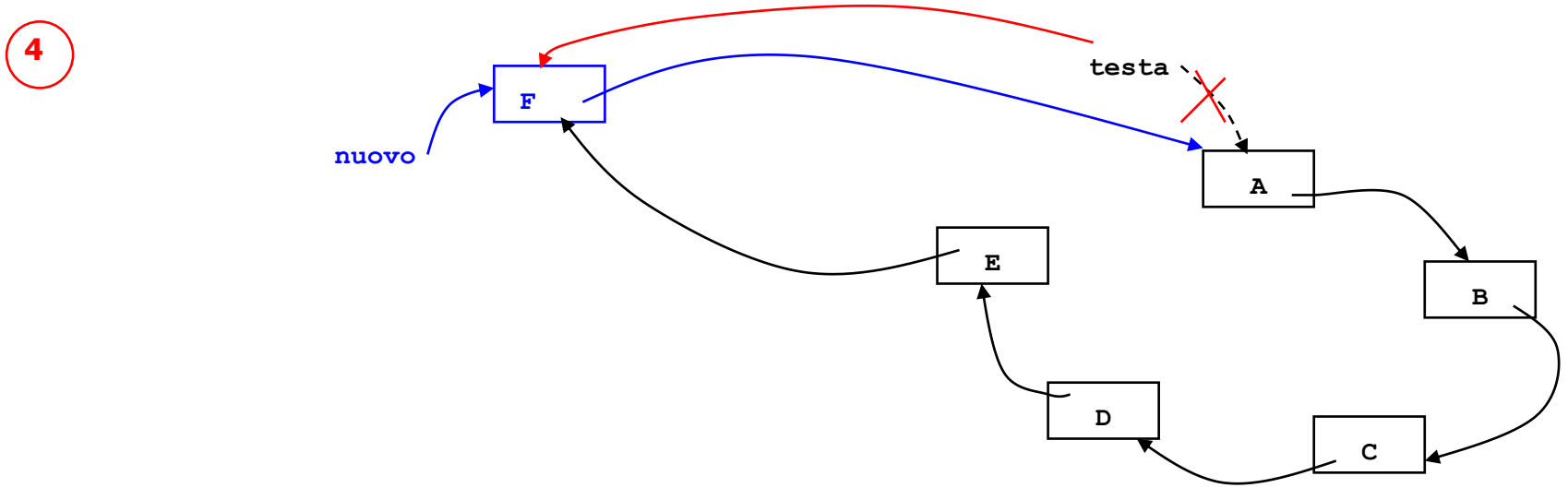
2



Poi dovremo collegare l'ultimo elemento della lista (considerando l'esempio la cella "E") alla nuova cella:



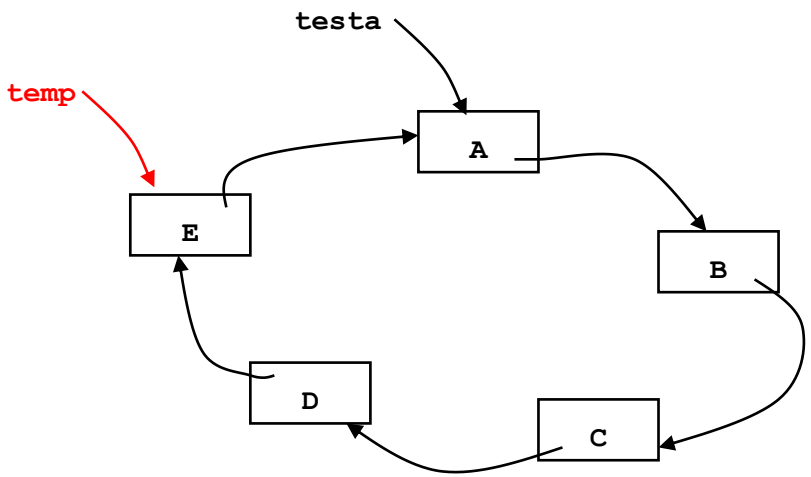
Infine spostare il riferimento della testa alla nuova cella:



Attenzione però perchè per poter fare l'operazione descritta al punto 3 avremo bisogno di un riferimento (puntatore) all'ultima cella!

Quindi prima di procedere con l'inserimento dovremo avere un riferimento all'ultima cella in lista, cioè quella cella dove il puntatore "next" si riferisce all'elemento puntato dalla testa

Quindi in modo da avere la seguente situazione:



Per fare questo dovremo cercare all'interno della lista la cella dove "next" ha lo stesso valore del puntatore testa:

```
temp = testa;
while (temp->next != testa)
      temp = temp->next;
```

Dopo il ciclo descritto il puntatore "temp" avrà il riferimento alla cella che precede la testa.

Quindi quanto descritto nel punto 3 ora sarà possibile grazie al puntatore temp:

```
nuovo = (struct cella*)malloc(sizeof(struct cella));
nuovo->next = testa;
temp->next = nuovo;
testa = nuovo;
```

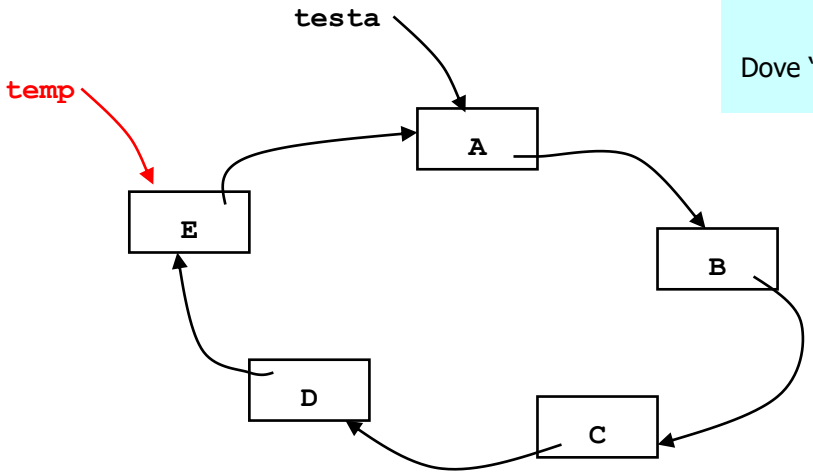
Riepilogando:

Cerchiamo la cella che precede quella in testa:

```
temp = testa;
while (temp->next != testa)
      temp = temp->next;
```

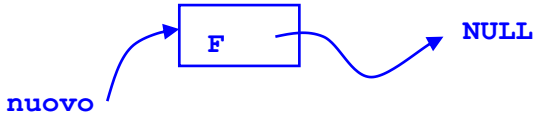
Dove "temp" è un puntatore di tipo struct cella

1



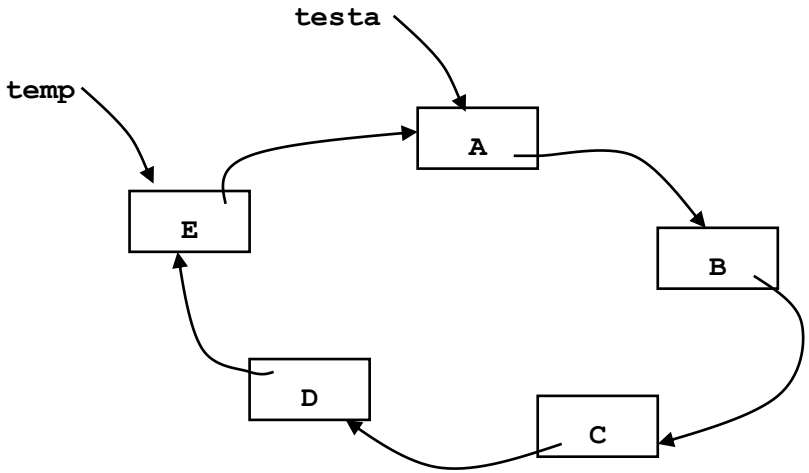
Attraverso la funzione malloc() allochiamo lo spazio necessario per mantenere in memoria la nuova cella da inserire all'interno della lista:

2



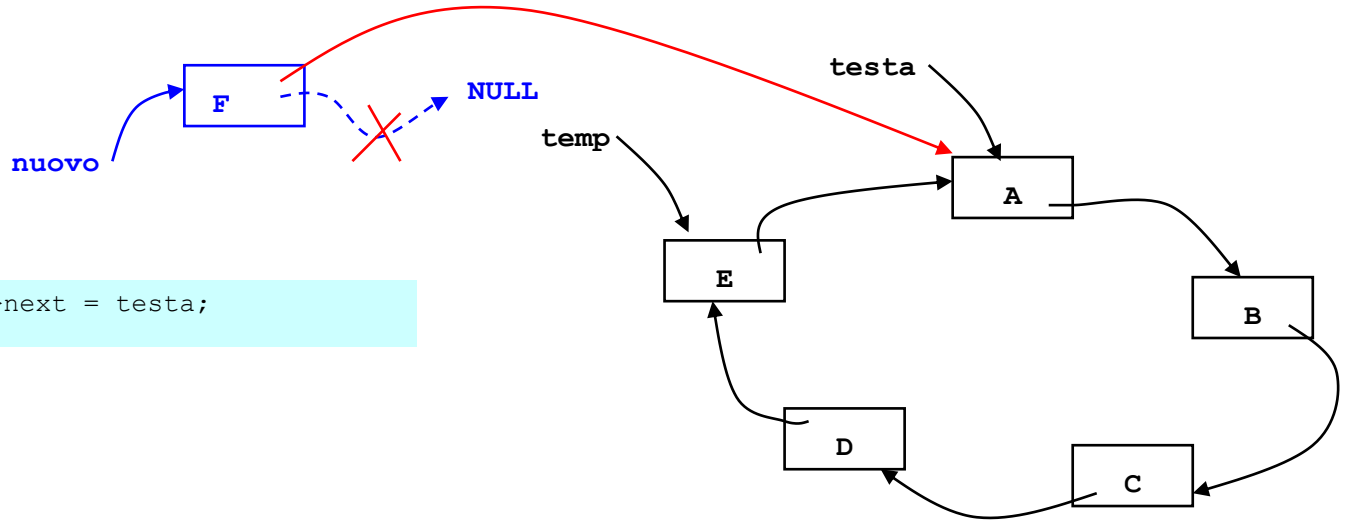

```
nuovo = (struct cella*)malloc(sizeof(struct cella));
```

Dove "nuovo" è un puntatore di tipo struct cella



La seconda operazione da compiere è collegare la nuova cella alla prima della lista, quindi fare in modo che l'elemento "next" della nuova cella prenda lo stesso valore di "testa":

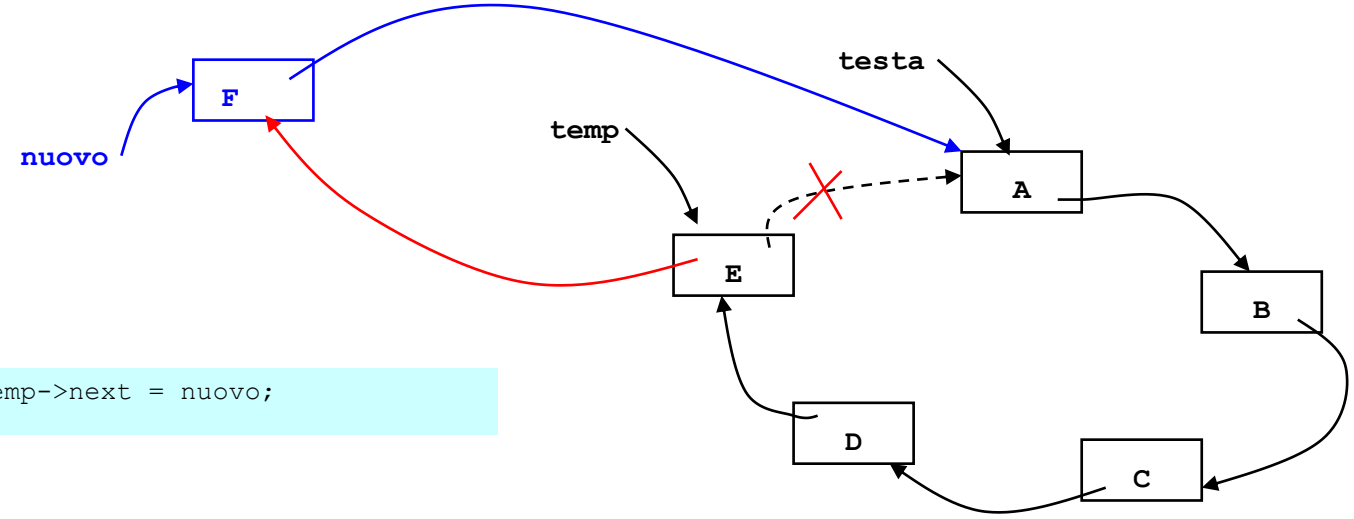
3




```
nuovo->next = testa;
```

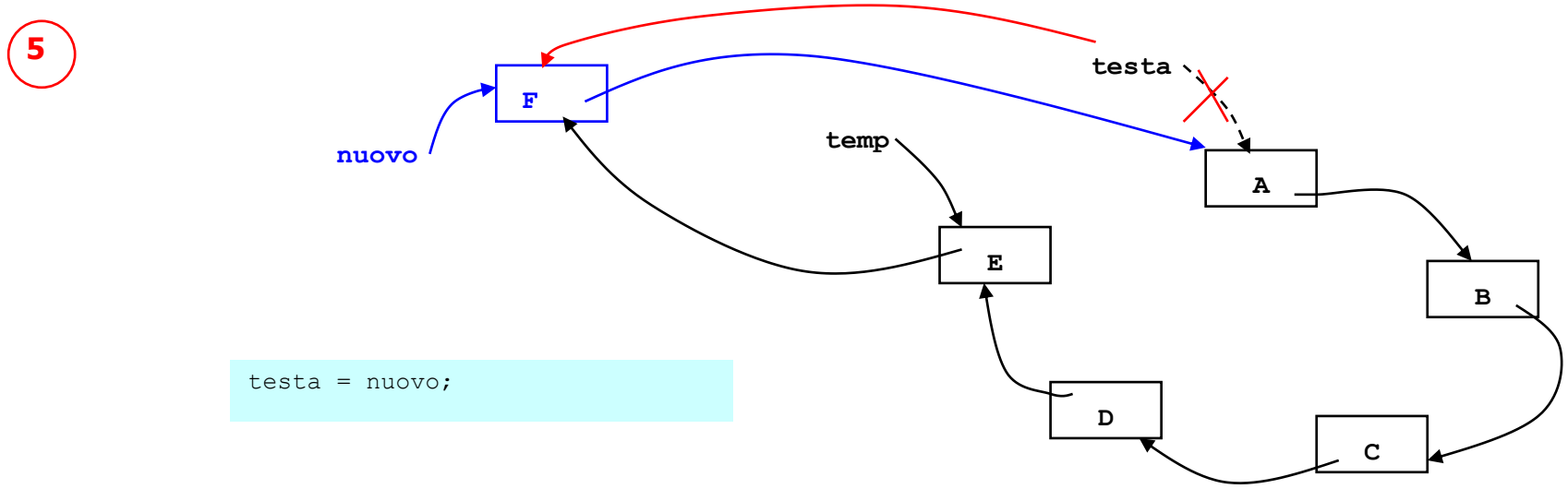
Poi dovremo collegare l'ultimo elemento della lista (considerando l'esempio la cella "E") alla nuova cella:

4




```
temp->next = nuovo;
```

Infine spostare il riferimento della testa alla nuova cella:



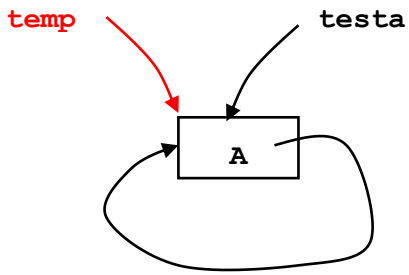

```
testa = nuovo;
```

Il codice completo:

```
temp = testa;
while (temp->next != testa)
      temp = temp->next;
nuovo = (struct cella*)malloc(sizeof(struct cella));
nuovo->next = testa;
temp->next = nuovo;
testa = nuovo;
```

Da notare che l'algoritmo descritto sopra funziona anche nel caso in cui la lista ha solo un elemento. In questo caso il puntatore "temp" avrà come riferimento la testa della lista:

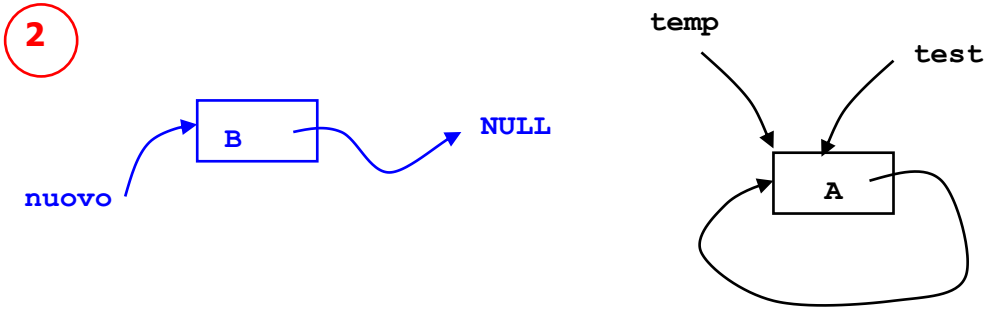
1




```
temp = testa;
while (temp->next != testa)
      temp = temp->next;
```

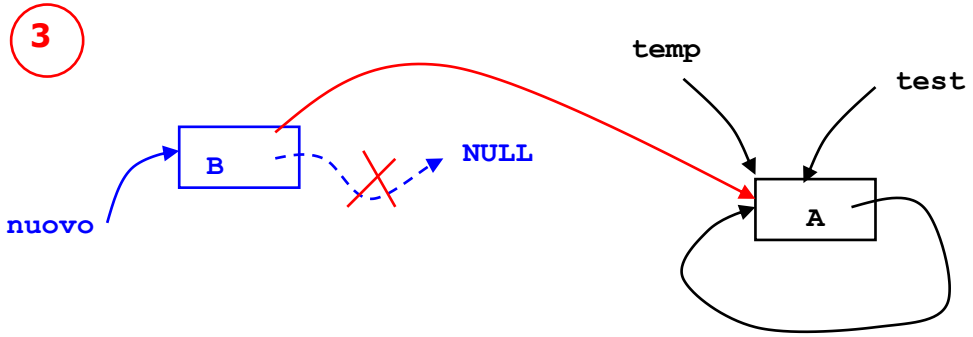
Non si entra nel ciclo e "temp" rimane uguale al valore di "testa"

2

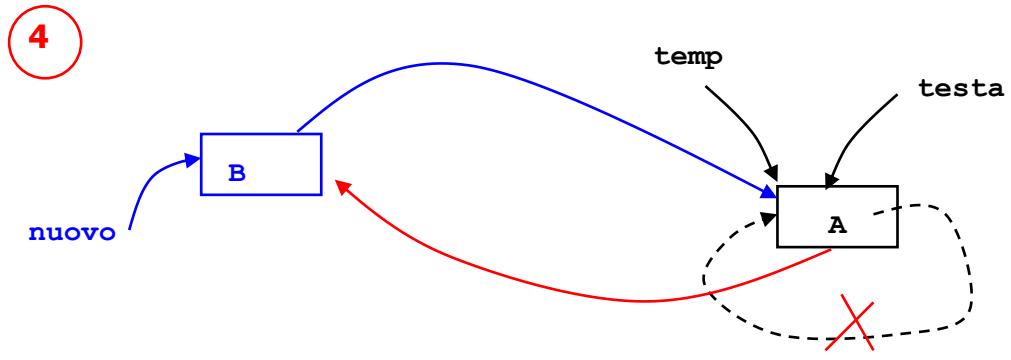



```
nuovo = (struct cella*)malloc(sizeof(struct cella));
```

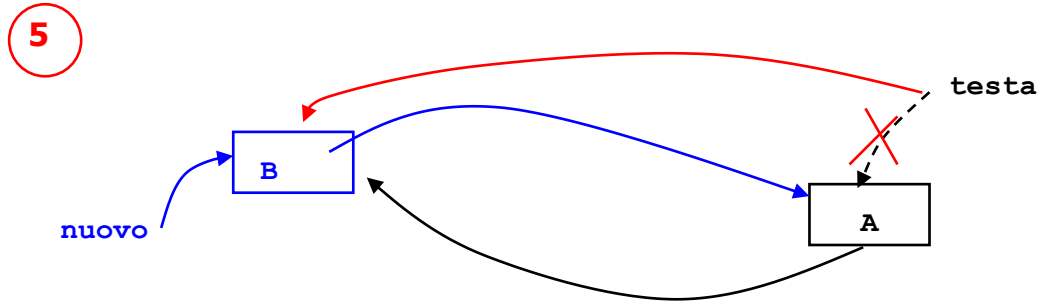
3




```
nuovo->next = testa;
```

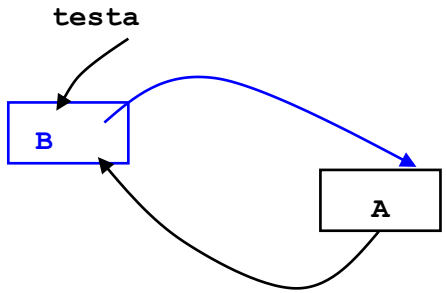



```
temp->next = nuovo;
```




```
testa = nuovo;
```

In modo da ottenere:

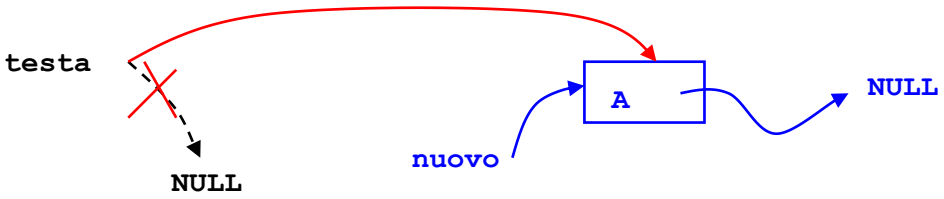


Attenzione perchè se la lista è vuota le operazioni descritte sopra non saranno tutte da eseguire, in particolare la condizioneall'interno del while: "temp->next != testa" NON potrà essere eseguita in quanto non esiste una cella all'interno della lista e quindi temp avrà il valore NULL!

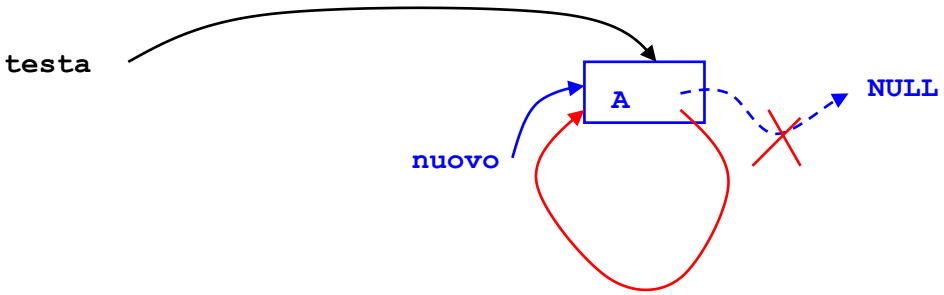
Se la lista è vuota le operazioni da fare saranno:

```
nuovo = (struct cella*)malloc(sizeof(struct cella));
```


```
testa = nuovo;
```




```
testa->next = testa;

oppure

nuovo->next = nuovo;
```



Quindi il codice completo per l'inserimento di una nuova cella all'interno di una lista circolare potrebbe essere:

```
nuovo = (struct cella*)malloc(sizeof(struct cella));
if (testa == NULL)
    temp = nuovo;
else
{
    temp = testa;
    while (temp->next != testa)
          temp = temp->next;
    nuovo->next = testa;
}

temp->next = nuovo;   //equivale a fare nuovo->next = nuovo se la lista è vuota!
testa = nuovo;
```

21.2 Stampa degli elementi il lista

Considerando la struttura dati e le variabili definite nel punto precedente, la stampa consiste nel accedere ad ogni elemento della lista partendo dal primo elemento in testa. La stampa degli elementi in una lista circolare è simile a quella di una lista semplice solo che invece di procedere con la stampa fino ad arrivare con il cursore (puntatore che ci permette di accedere di volta in volta alle varie celle) a NULL, in questo caso si procederà fino all'ultima cella ovvero alla cella che come riferimento alla cella successiva (next) ha lo stesso valore del puntatore alla testa:

```
if (testa != NULL)
{
    temp = testa;
    do
    {
          printf("%c",temp->valore);
          temp = temp->next;
    }
    while (temp != testa);
}
```

21.3 Ricerca di un elemento in lista

La ricerca è simile alla stampa:

```
trovato = 0;
if (testa != NULL)
{
    temp = testa;
    do
    {
          if (temp->valore == valoreDaCercare)
          {
                trovato = 1;
                break;
          }
          temp = temp->next;
    }
    while (temp != testa);
}
if (trovato)
     printf("Il valore cercato è presente in lista!");
else
     printf("valore NON trovato");
```

21.4 Numero di occorrenze di un elemento in lista

Questa operazione è simile a quella del punto precedente (in questo caso bisogna però contare quante volte un elemento è ripetuto all'interno della lista):

```
conta = 0;
if (testa != NULL)
{
    temp = testa;
    do
    {
        if (temp->valore == valoreDaCercare)
            conta++;
        temp = temp->next;
    }
    while (temp != testa);
}
printf("L'elemento cercato è presente %d volte!",conta);
```

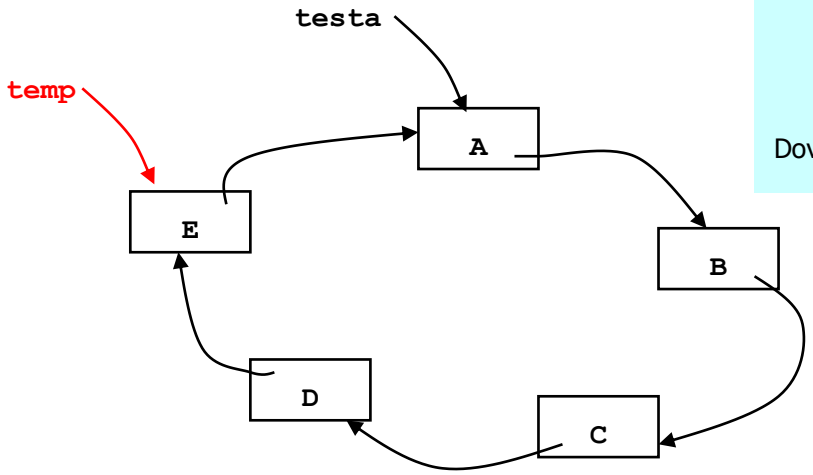
21.5 Numero di elementi in lista

```
conta = 0;
if (testa != NULL)
{
    temp = testa;
    do
    {
        conta++;
        temp = temp->next;
    }
    while (temp != testa);
}
printf("La lista ha %d elementi!",conta);
```

21.6 Cancellazione dell'elemento in testa alla lista

Prima di cancellare l'elemento in testa dovremo trovare il riferimento all'ultimo elemento in lista, come visto per l'inserimento in modo tale da poter mantenere la logica della lista circolare anche dopo l'eliminazione della cella in testa alla lista:

1

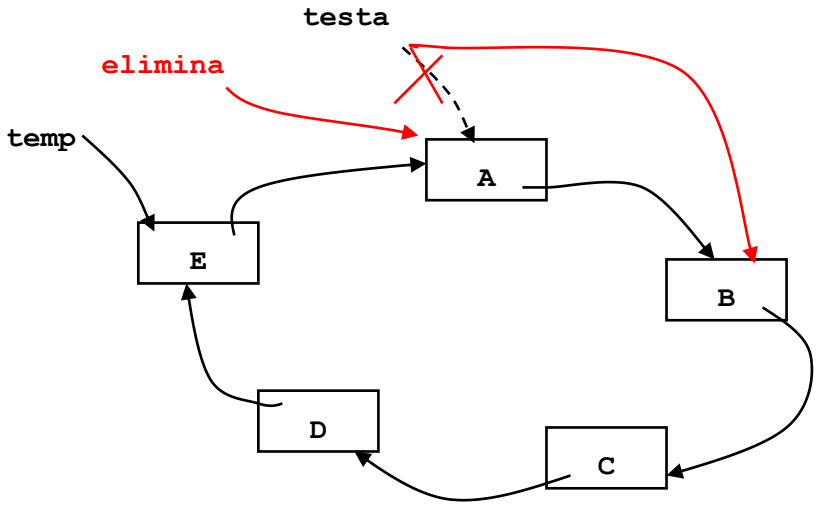



```
temp = testa;
while (temp->next != testa)
      temp = temp->next;
```

Dove "temp" è un puntatore di tipo struct cella

Successivamente la testa dovrà puntare all'elemento successivo in modo da non perdere il riferimento alla lista, prima però dovremo tener traccia dell'elemento da eliminare attraverso un nuovo puntatore:

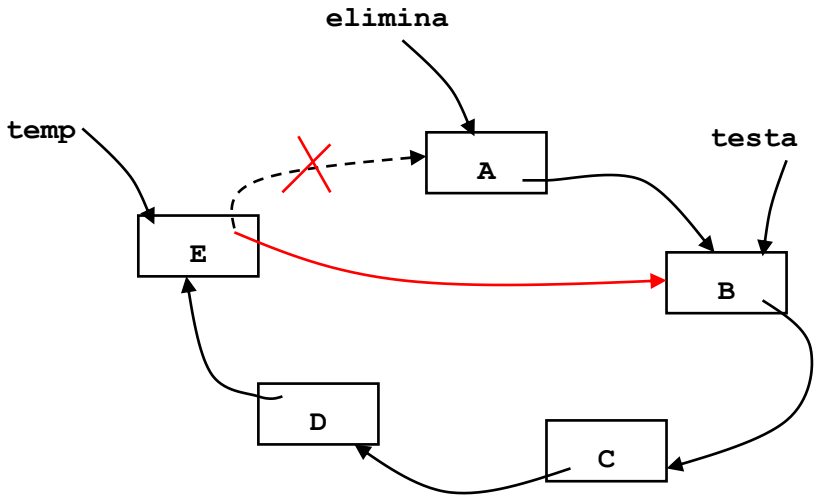
2




```
elimina = testa;
testa = testa->next;
```

Poi l'elemento in fondo alla lista dovrà puntare alla nuova testa della lista:

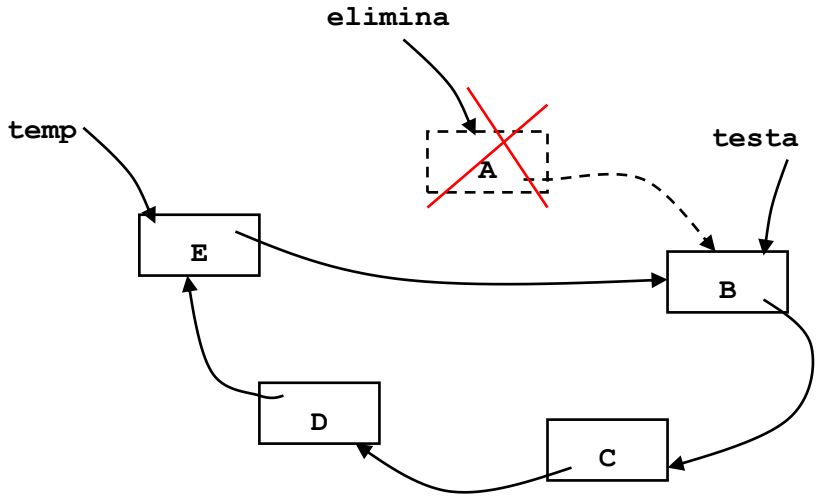
3




```
temp->next = testa;
```

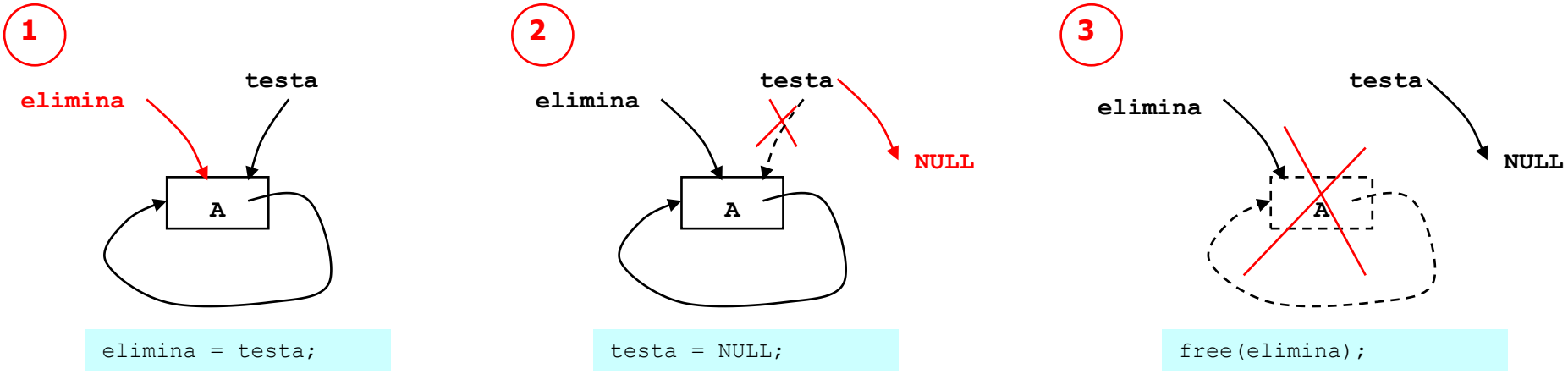
Come ultima operazione attraverso la funzione free() verrà liberata la memoria dalla cella da eliminare:

4




```
free(elimina);
```

Attenzione! Se la lista ha solo un elemento le operazioni cambiano!




```
elimina = testa;
```

```
testa = NULL;
```

```
free(elimina);
```

Riassumendo il codice necessario per eliminare l'elemento in testa alla lista:

```
elimina = testa;
temp = testa;
while (temp->next != testa)
      temp = temp->next;

if (temp == testa) //esiste un solo elemento in lista
      testa = NULL;
else
{
      testa = testa->next;
      temp->next = testa;
}

free(elimina);
```

21.7 Cancellazione di un elemento all'interno della lista circolare

Le operazioni sono simili alla cancellazione di un elemento per una lista semplice eccetto il caso in cui l'elemento da eliminare è in testa alla lista (caso analizzato nel paragrafo precedente).

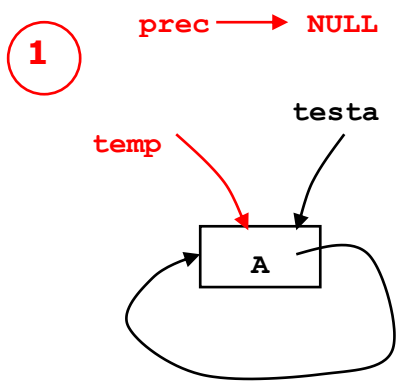
Dovremo quindi scorrere tutta la lista fino a trovare l'elemento da eliminare o fino ad aver compiuto un giro completo all'interno della lista. Nell'analizzare la lista dovremo tener traccia, attraverso un puntatore, anche dell'elemento che precede quello considerato durante l'analisi (ad esempio il puntatore chiamato "prec" come per le liste semplici)

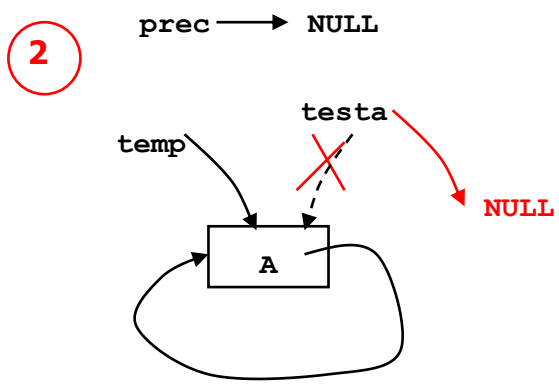
Una volta individuata la cella da eliminare si procederà come per le liste semplici eccetto il caso in cui l'elemento sia posizionato in testa alla lista.

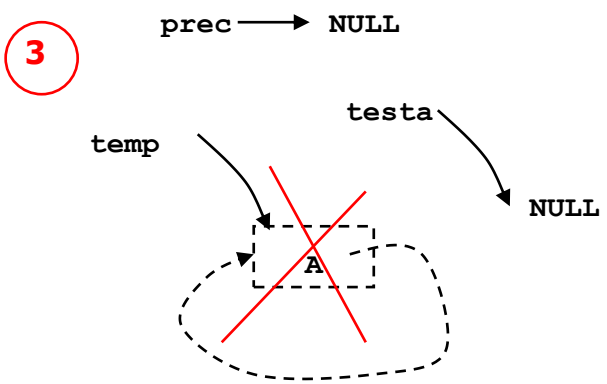
```
if (testa != NULL) //la lista non è vuota
{
     prec = NULL;
     temp = testa;
     do
     {
            if (temp->valore == elementoDaEliminare) //temp è il riferimento all'elemento da eliminare
            {
                  if (prec == NULL) //l'elemento da eliminare si trova in testa
                  {
                        if (temp->next == testa) //l'elemento da eliminare è l'unico in lista
                              testa = NULL;
                        else
                        {
                              prec = testa;
                              /*per individuare il precedente nel caso in cui l'elemento da
                              eliminare sia intesta*/
                               while (prec->next != testa)
                                    prec = prec->next;
                              testa = testa->next;
                              prec->next = temp->next;
                        }
                  }
                  else
                        prec->next = temp->next;
                  free(temp);
                  break;
            }
            prec = temp;
```

```
                temp = temp->next;
            }
            while (temp != testa);
        }
```

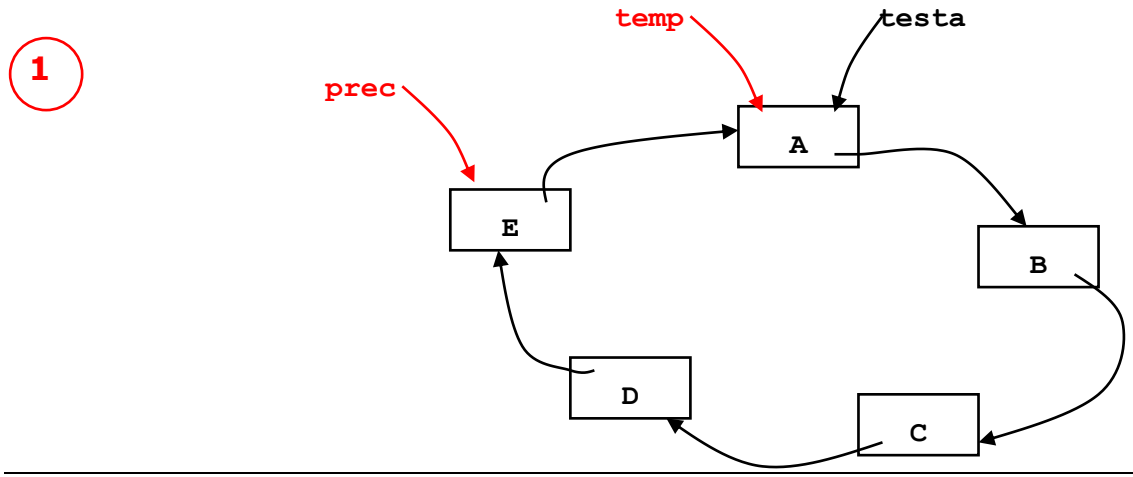
L'elemento da eliminare è l'unico in lista (come visto nel paragrafo precedente):







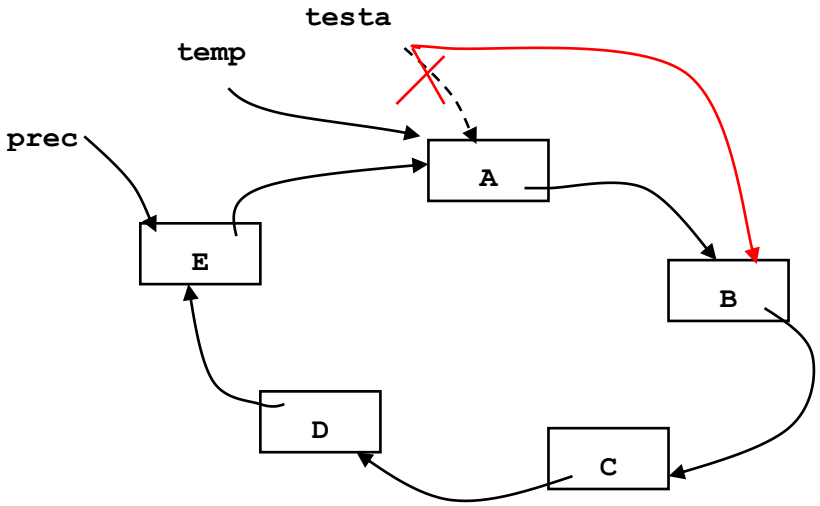
Se l'elemento da eliminare è in testa e la lista ha più di un elemento:




```
        prec = testa;
        while (prec->next != testa)
              prec = prec->next;
```

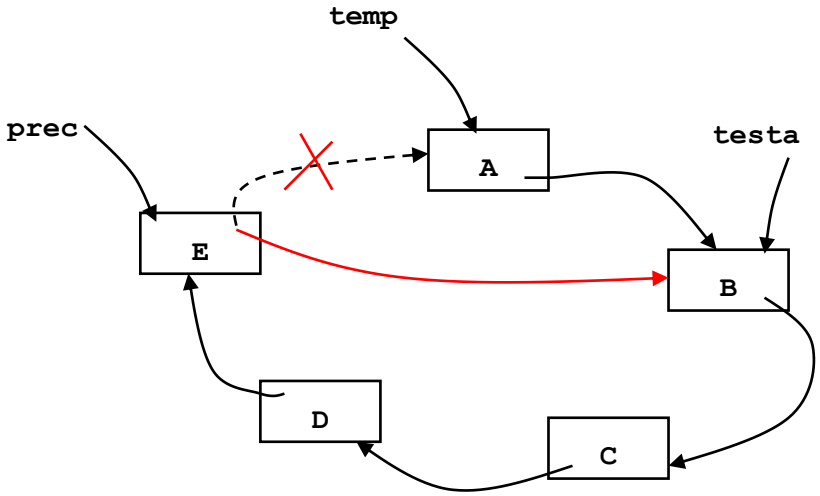
"prec" viene individuato attraverso uno scorrimento completo della lista

2



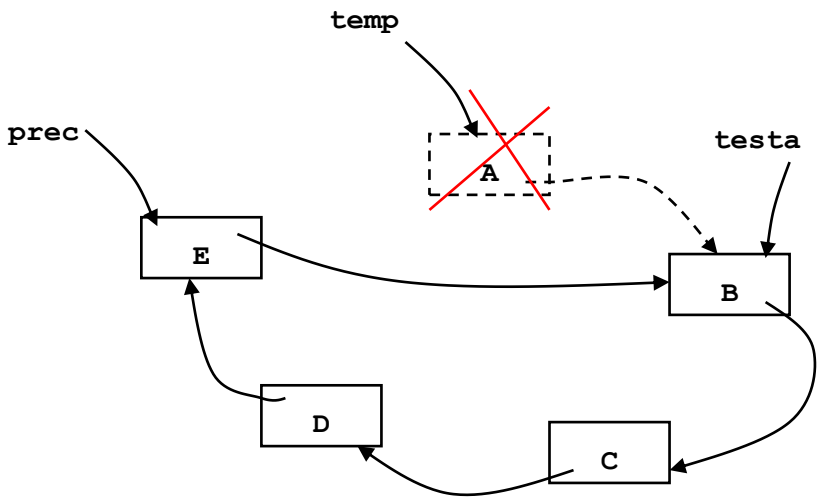

```
testa = testa->next;
```

3




```
prec->next = temp->next;
```

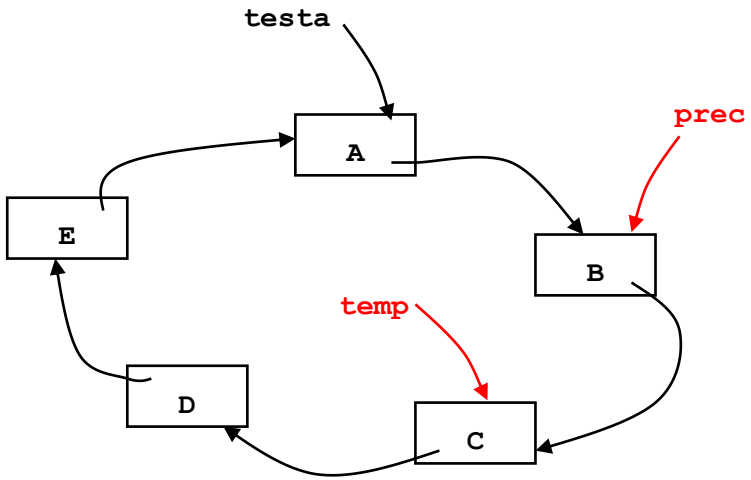
4



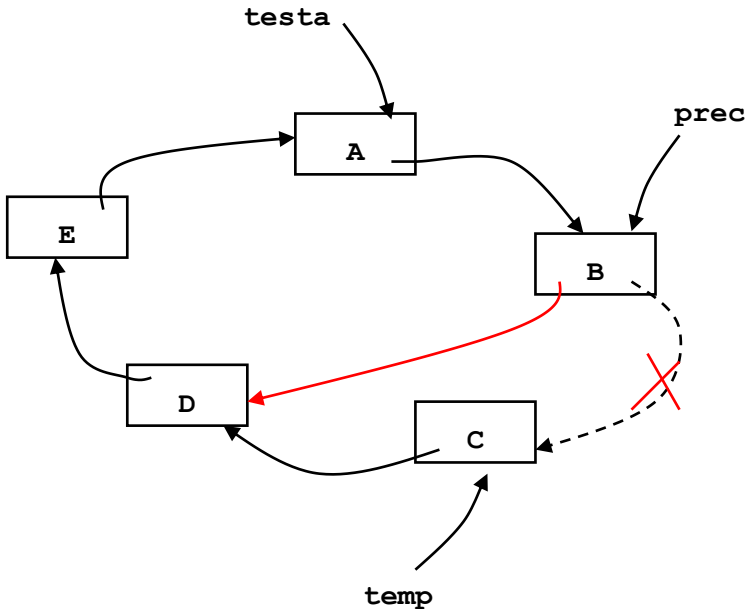

```
free(temp);
```

Se l'elemento da eliminare è all'interno della lista:

1

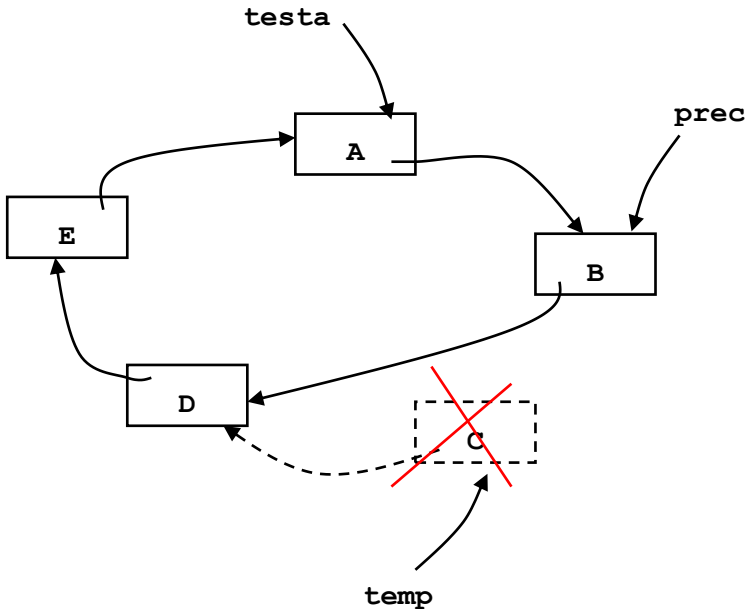


2




```
prec->next = temp->next;
```

3




```
free(temp);
```