PROGRAMMAZIONE 2: SPERIMENTAZIONI

Lezione 6 (parte 3) – Algoritmi ricorsivi sulle liste in C

Agenda

- Introduzione
- Algoritmi ricorsivi sulle liste
  - Creare una lista
  - Visualizzare una lista
  - Contare gli elementi (nodi) di una lista
  - Sommare gli elementi (nodi) di una lista
  - Cancellare una lista
  - Ricercare un elemento (nodo)
  - Cancellare un elemento (nodo)

Video lezione disponibili su YouTube: https://youtu.be/Y07EHoCRQZ8

Introduzione

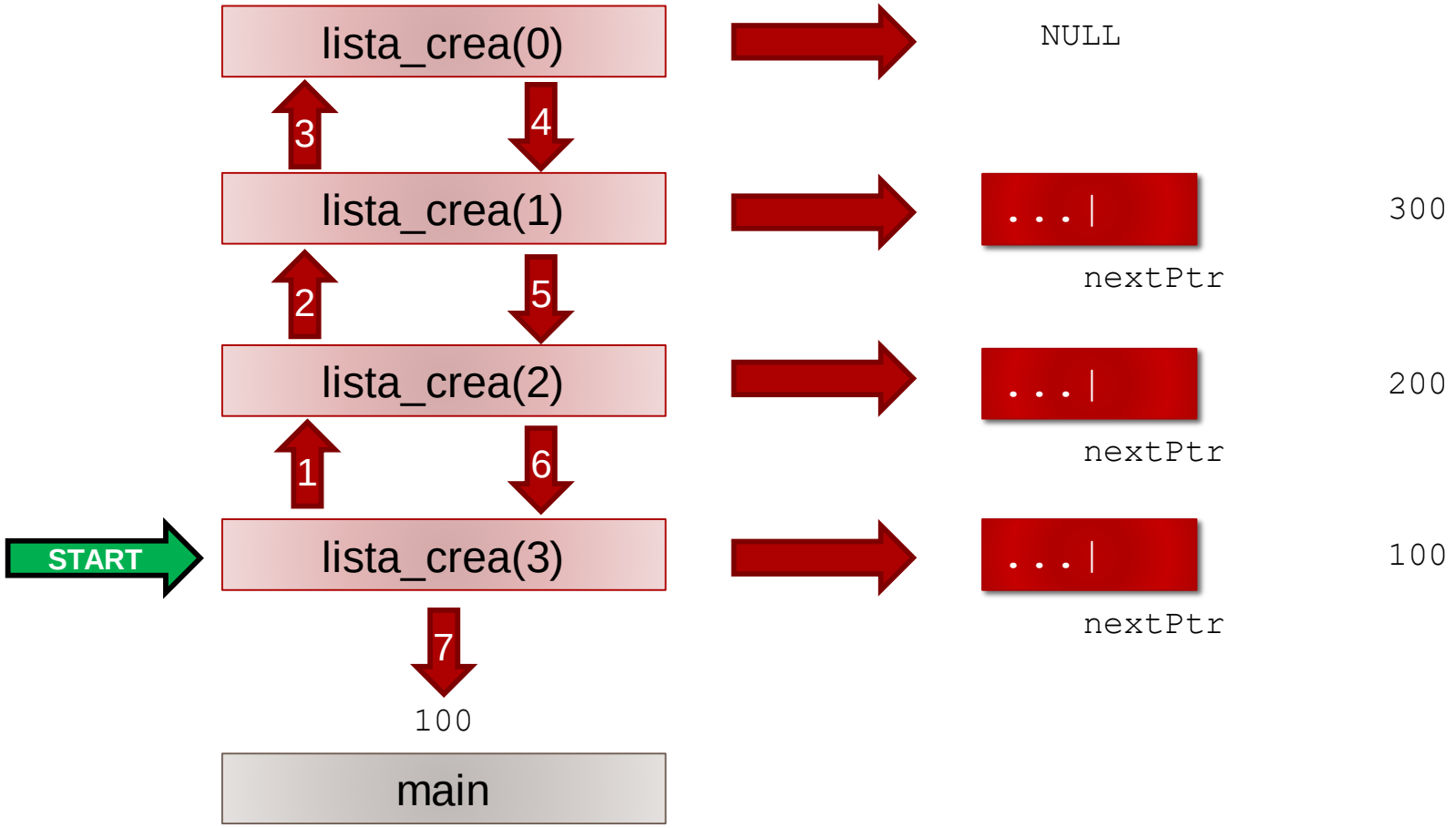
- La ricorsione risulta particolarmente utile sulle liste collegate.
- Questa utilità è dovuta al fatto che le liste stesse possono essere definite in modo ricorsivo (una lista è una lista vuota oppure un elemento (nodo) seguito da un'altra lista).
- Nel seguito della lezione verranno analizzati alcuni algoritmi che utilizzano la ricorsione sulle liste.

Creare una lista

es1.c

```
37  Lista * lista_crea(int n)
38  {
39    Lista *headPtr;
40
41    // caso base
42    if (n==0)
43    {
44      return NULL;
45    }
46    else // caso ricorsivo
47    {
48      headPtr = malloc(sizeof(Lista));
49      headPtr->data = 1 + rand() % 6; // genera numeri casuali tra 1 e 6
50      headPtr->nextPtr = lista_crea(n-1);
51      return headPtr;
52    }
53  }
```

Creare una lista

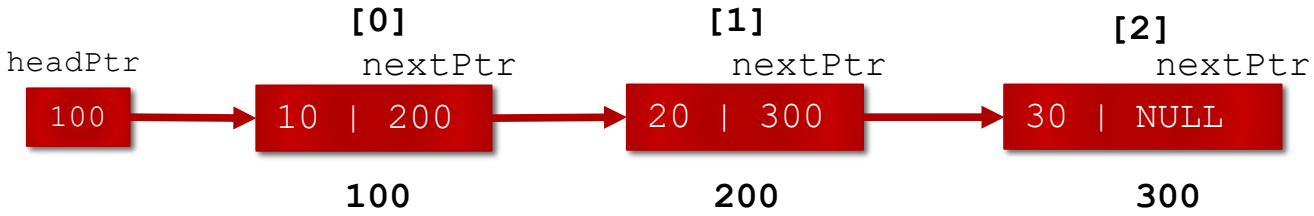


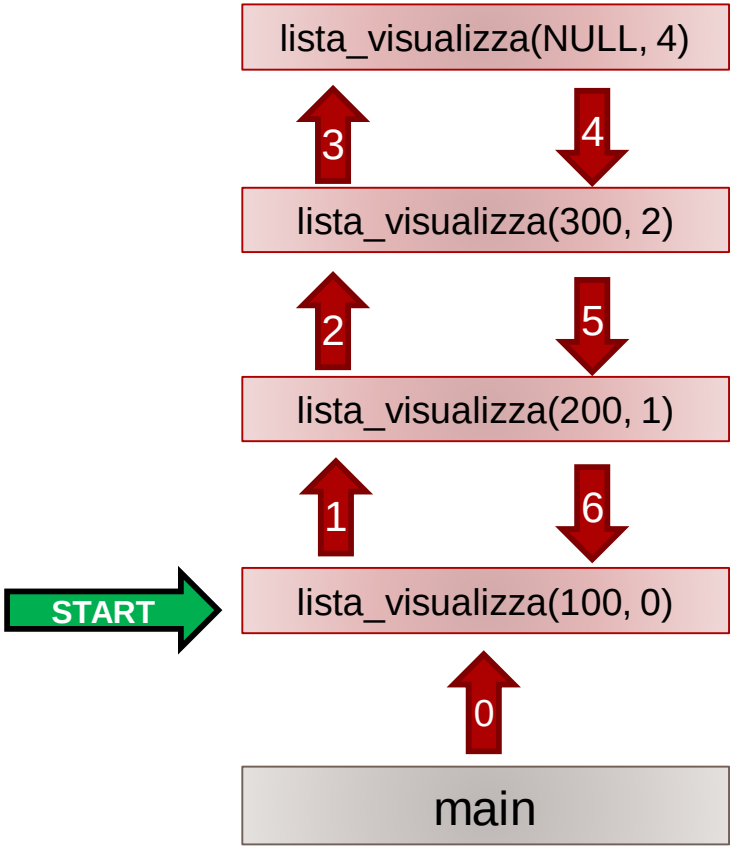
Visualizzare una lista

es1.c

```
56  void lista_visualizza(Lista *headPtr, int p)
57  {
58    // caso base
59    if (headPtr==NULL)
60    {
61      return;
62    }
63    else // caso ricorsivo
64    {
65      printf("Nodo [%d]: %d \n", p , headPtr->data);
66      return lista_visualizza(headPtr->nextPtr, p+1);
67    }
68  }
```

Visualizzare una lista



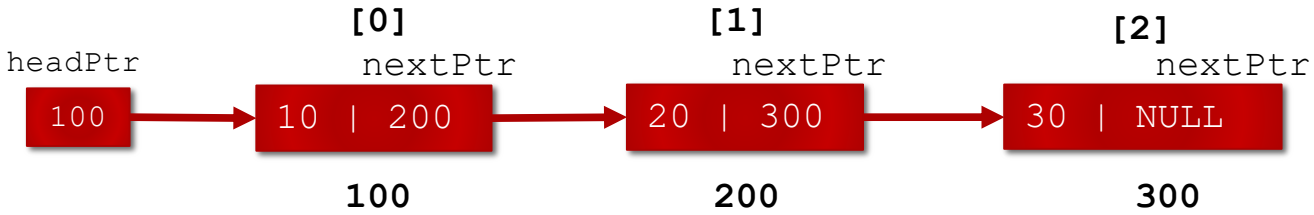


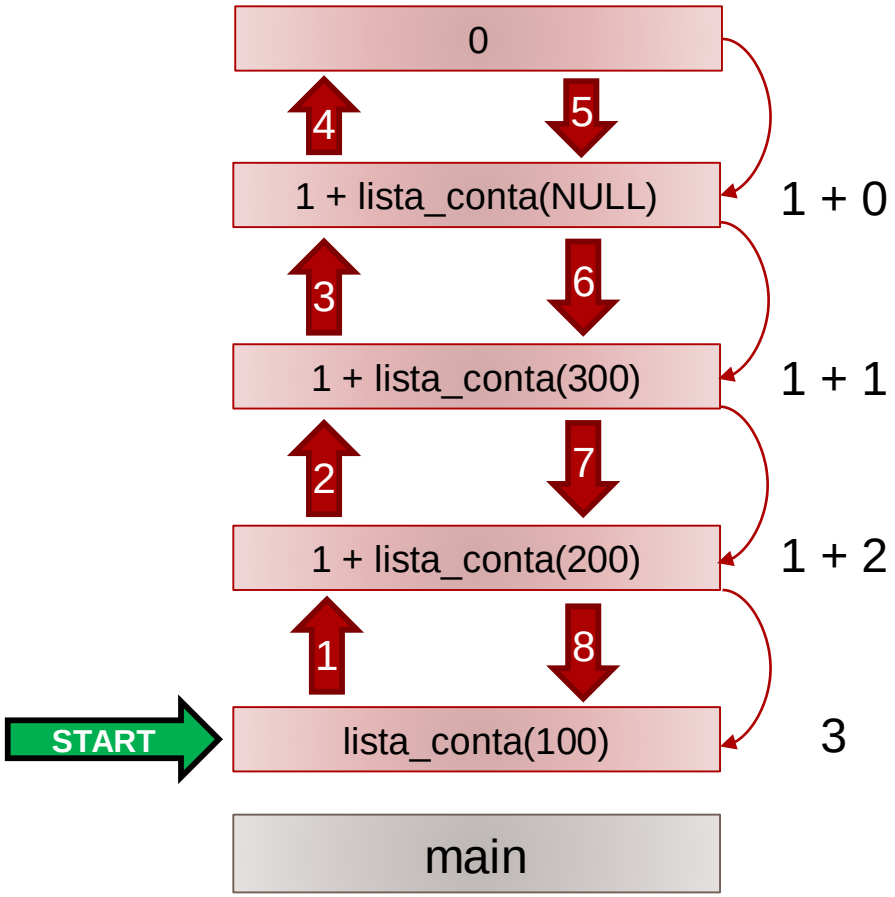
👉Contare gli elementi di una lista

es2.c

```
76  int lista_conta(Lista *headPtr)
77  {
78    // caso base
79    if (headPtr==NULL)
80    {
81      return 0;
82    }
83    else
84    {
85      return 1 + lista_conta(headPtr->nextPtr);
86    }
87  }
```

Contare gli elementi di una lista



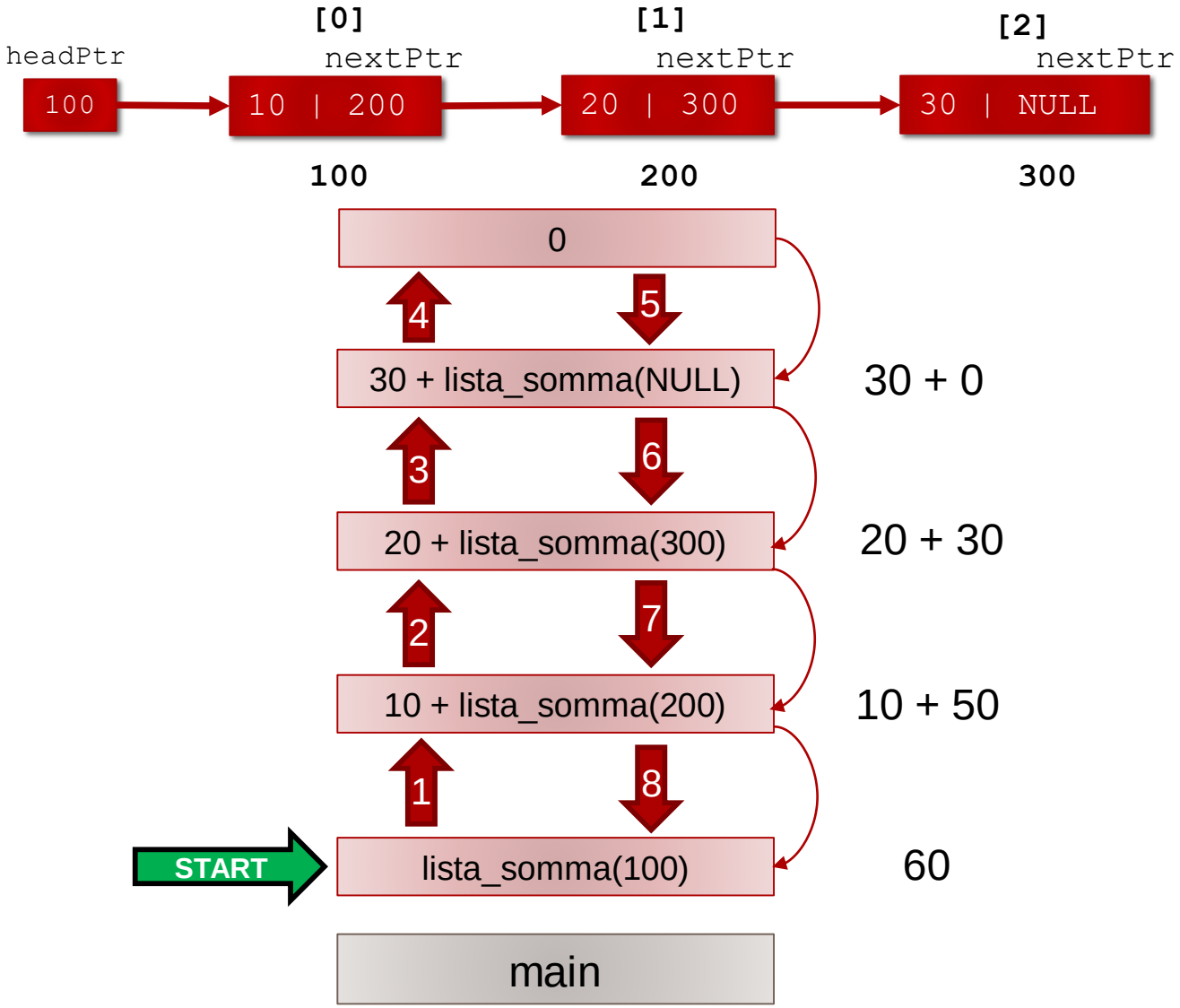


Sommare gli elementi di una lista

es3.c

```
77  int lista_somma(Lista *headPtr)
78  {
79    // caso base
80    if (headPtr==NULL)
81    {
82      return 0;
83    }
84    else // caso ricorsivo
85    {
86      return headPtr->data + lista_somma(headPtr->nextPtr);
87    }
88  }
```

Sommare gli elementi di una lista

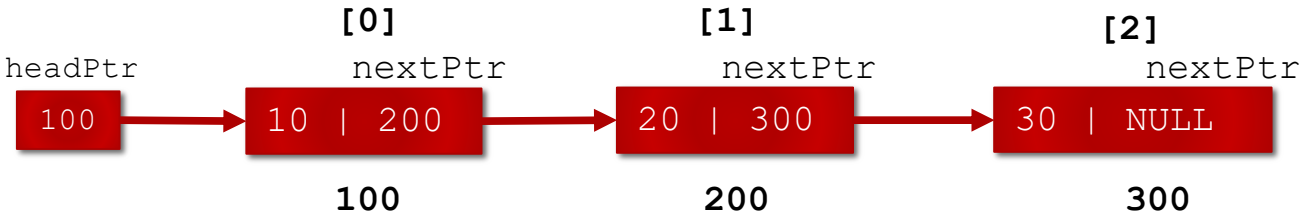


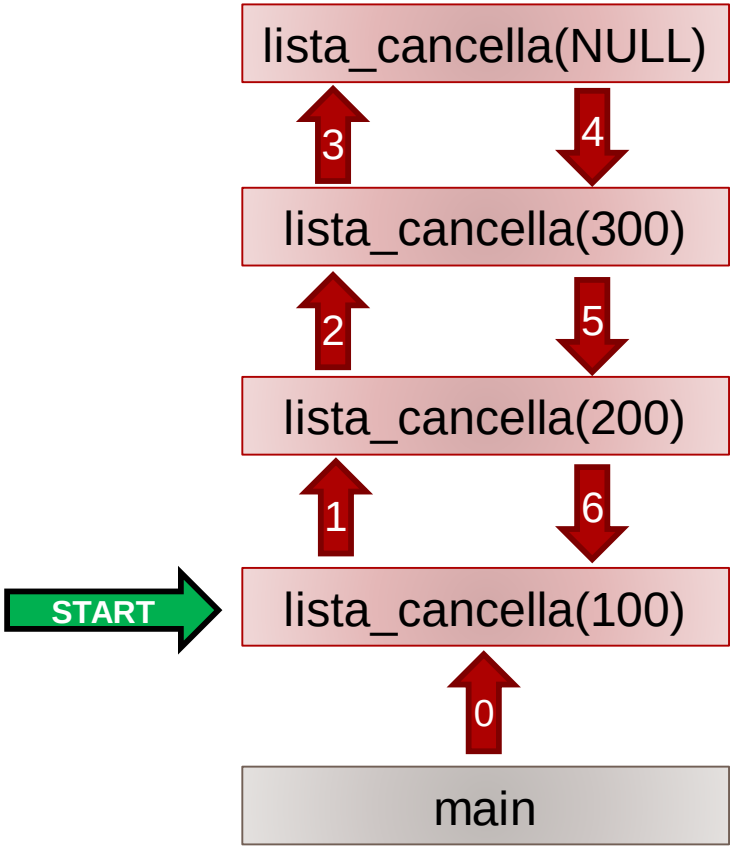
👉Cancellare una lista

es4.c

```
74  void lista_cancella(Lista *headPtr)
75  {
76    // caso base
77    if (headPtr==NULL)
78    {
79      return;
80    }
81      lista_cancella(headPtr->nextPtr);
82      free(headPtr);
83  }
```

Cancellare una lista



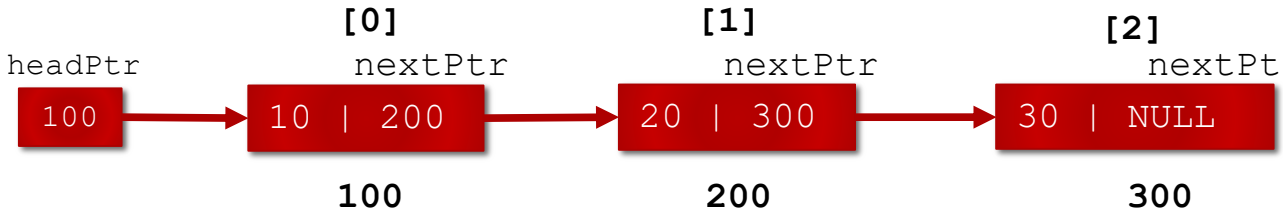


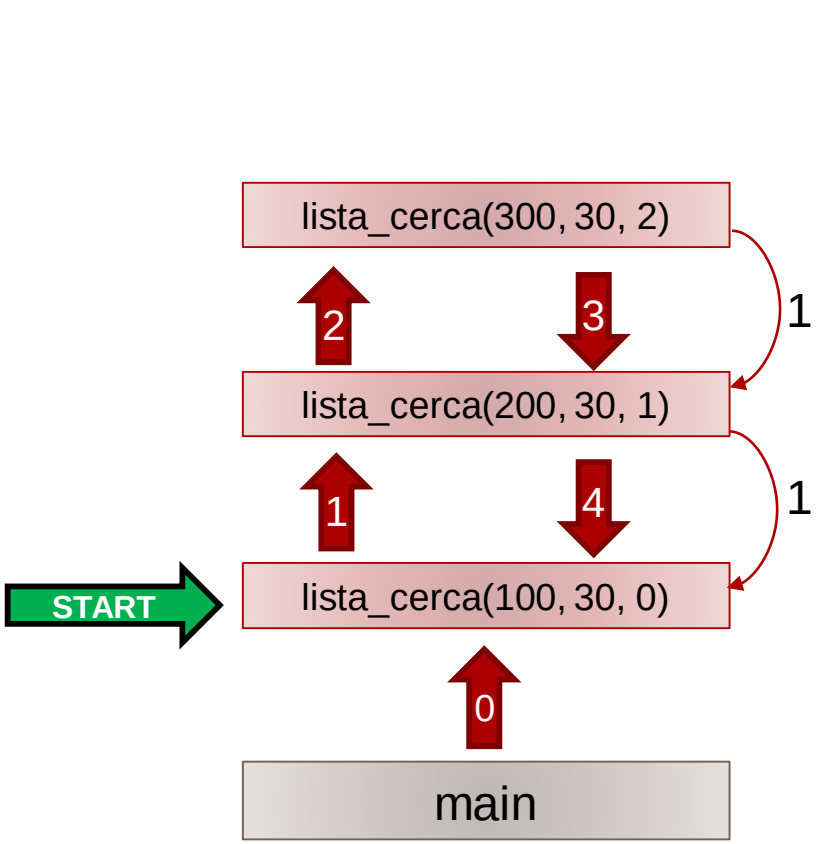
Ricercare un elemento (nodo)

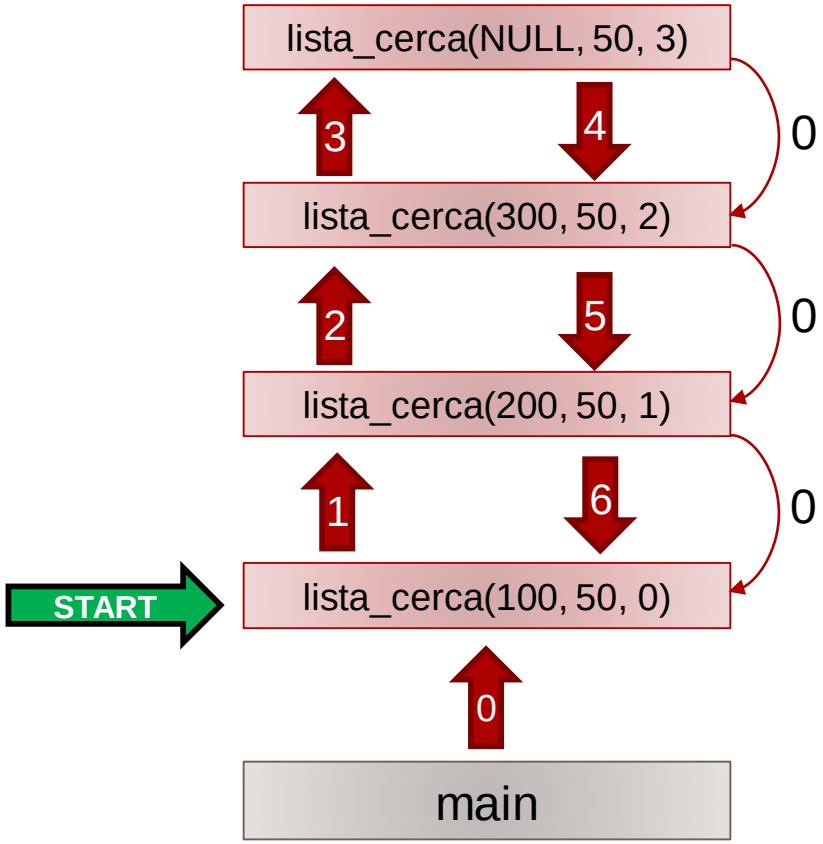
es5.c

```
86  int lista_cerca(Lista *headPtr, int x, int *p)
87  {
88    // caso base
89    if (headPtr==NULL)
90    {
91       *p = -1;
92       return 0; // falso
93    }
94
95    if (headPtr->data == x)
96      return 1; // vero
97
98    // caso ricorsivo
99    (*p)++; // *p = *p + 1
100   return lista_cerca(headPtr->nextPtr, x, p);
101 }
```

Ricercare un elemento (nodo)





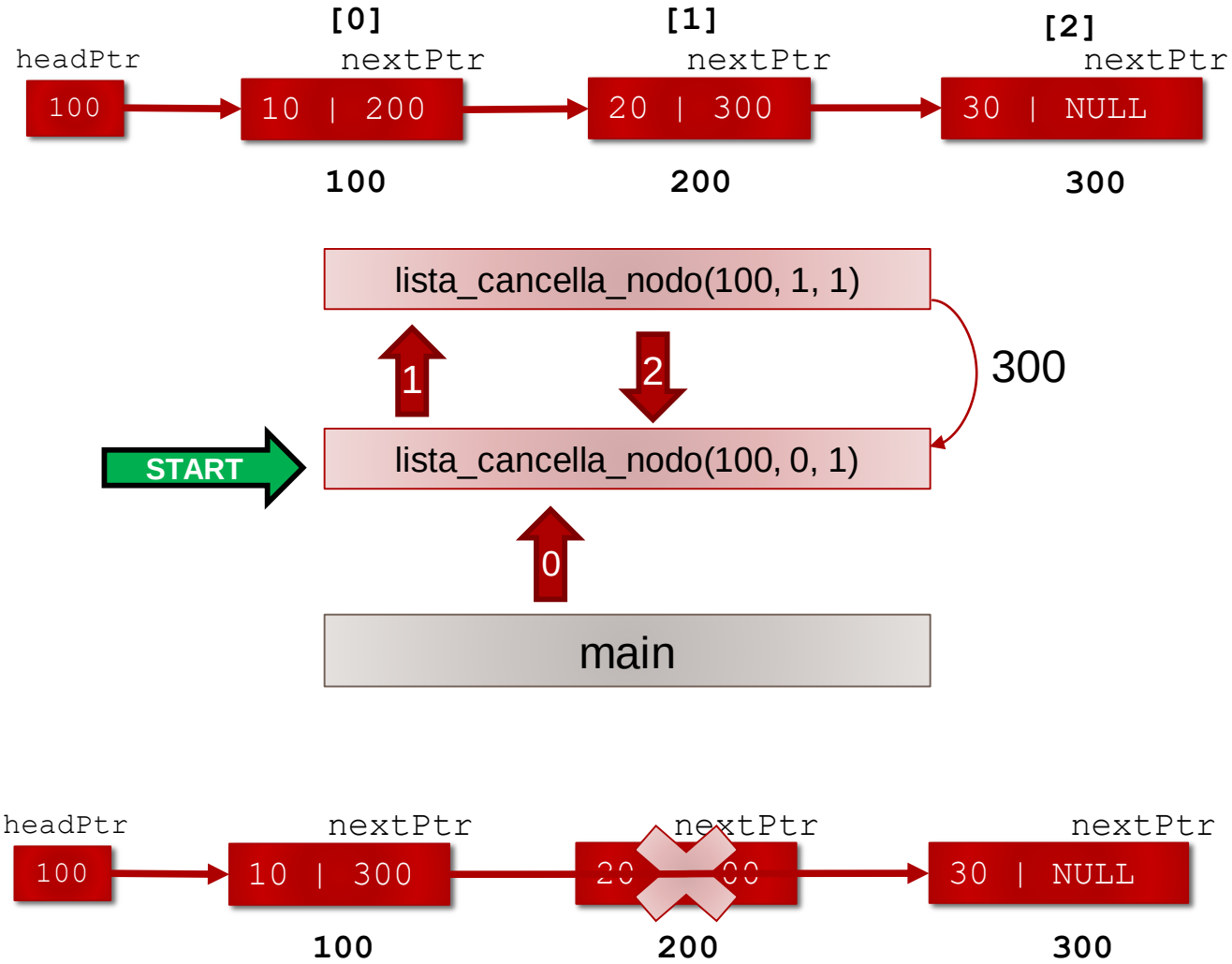


Cancellare un elemento (nodo)

es6.c

```
84  // p è la posizione del nodo corrente (partendo da 0)
85  // k è la posizione del nodo che si desidera cancellare
86  Lista * lista_cancella_nodo(Lista *headPtr, int p, int k)
87  {
88    // se la lista è vuota
89    if (headPtr == NULL)
90      return NULL;
91
92    if (p == k)
93    {
94        Lista *newPtr = headPtr->nextPtr;
95        free(headPtr);
96        return newPtr;
97    }
98
99     headPtr->nextPtr = lista_cancella_nodo(headPtr->nextPtr, p+1, k);
100    return headPtr;
101 }
```

Cancellare un elemento (nodo)



FINE PRESENTAZIONE