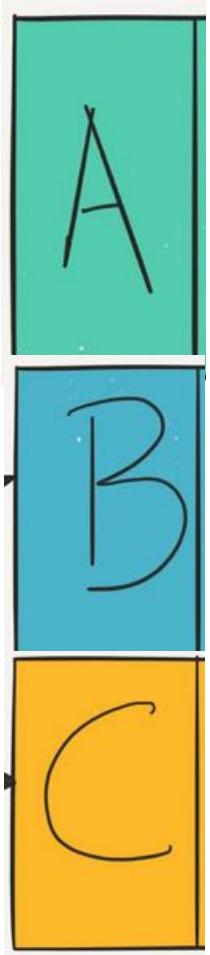
Liste concatenate semplici

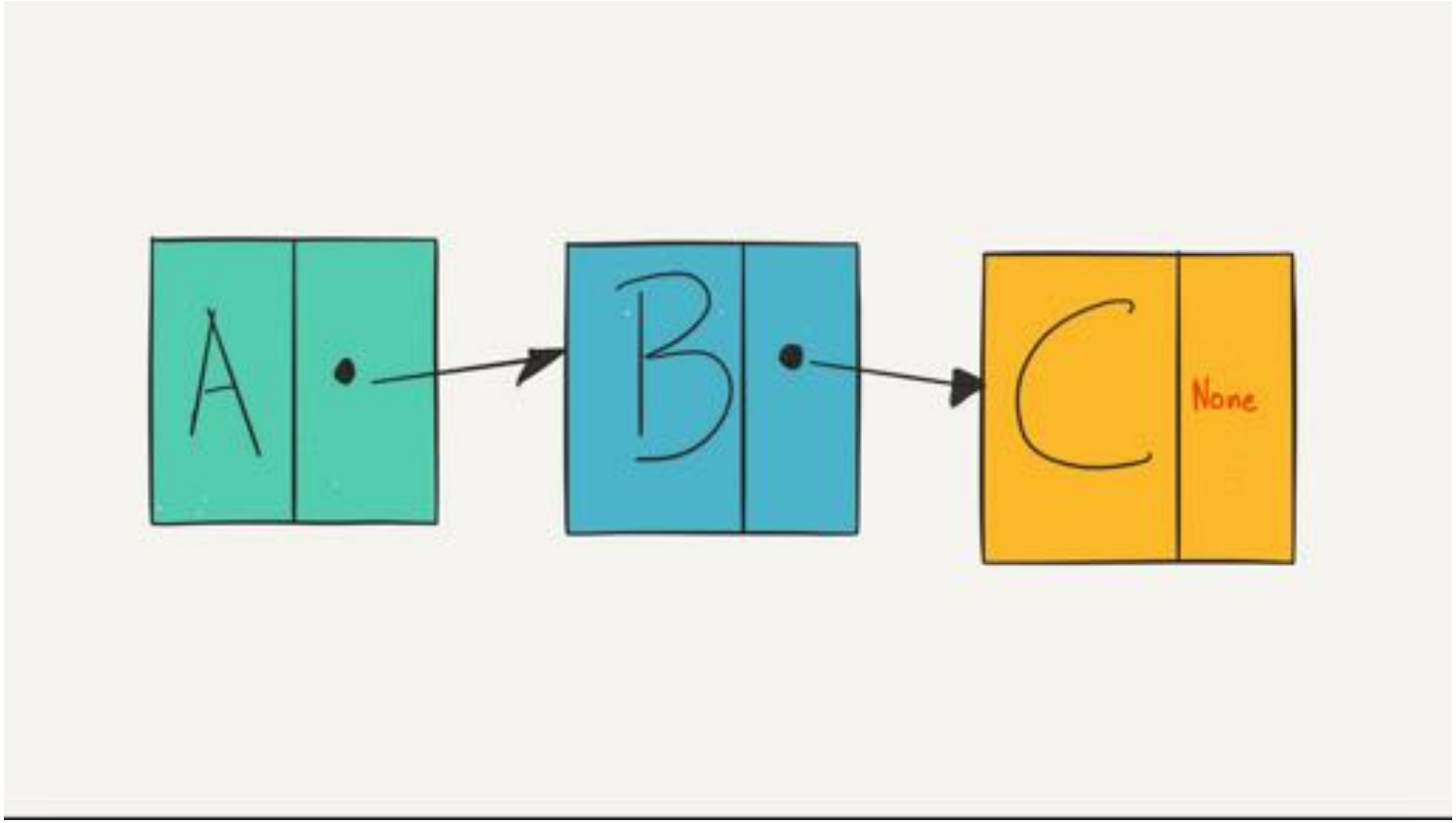
Obiettivi:

- Presentare la realizzazione collegata (puntatori a strutture) di liste semplici

Dai vettori … alle liste

```
char V[3]={'A','B','C'};
```





IL CONCETTO DI LISTA

Una lista semplice è una sequenza (*multi-insieme finito e ordinato*) **di elementi dello *stesso tipo***

Multi-insieme: insieme in cui un medesimo elemento può comparire più volte

Notazione spesso usata: $\mathtt{L = [el_1, el_2, \ldots, el_N]}$

[‘a’ , ‘b’ , ‘c’] denota la lista dei caratteri ‘a’ , ‘b’ , ‘c’
[5, 8, 5, 21, 8] denota una lista di 5 interi

Come ogni ***ADT***, il tipo **lista** è definito in termini di:

- ***dominio*** dei suoi elementi (dominio-base)
- operazioni di ***costruzione*** sul tipo lista, operazioni di ***selezione*** sul tipo lista
- ***predicati***

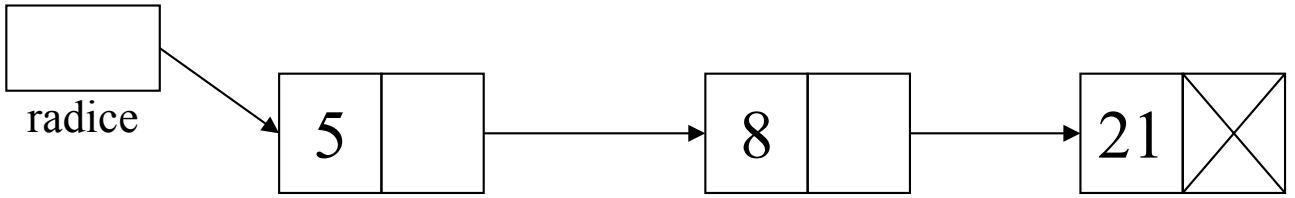
RAPPRESENTAZIONE COLLEGATA

Le liste sono realizzabili anche con array (rappresentazione sequenziale): la successione degli elementi della lista è realizzata dall'adiacenza delle locazioni; non la trattiamo.

Nella **rappresentazione collegata**, a ogni elemento si associa l'indirizzo ***(puntatore)*** che individua la posizione dell'elemento successivo

NOTAZIONE GRAFICA
- elementi della lista come ***nodi***
- riferimenti (indici) come ***archi***

Esempio:
[5, 8, 21]

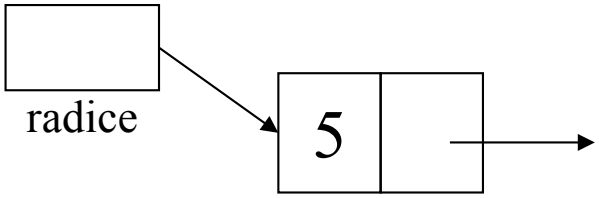


Liste - RAPPRESENTAZIONE COLLEGATA

IMPLEMENTAZIONE MEDIANTE PUNTATORI

Ciascun nodo della lista è una struttura di due campi:

- ***valore*** dell'elemento
- un ***puntatore*** nodo successivo lista (**NULL** se ultimo elemento)



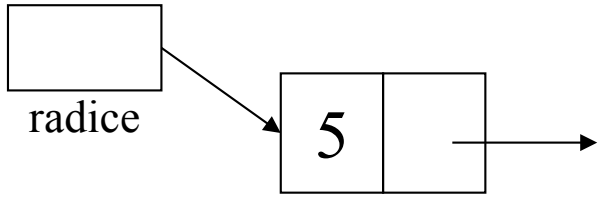
•Cosa vorremmo fare (ma in C non si può … non posso usare `item` prima di averlo definito):

```
typedef struct {
        int value;
        item *next;} item;      //ERRORE!!!!!
typedef item *list;
list radice;
```

Liste - RAPPRESENTAZIONE COLLEGATA

Ciascun nodo della lista è una struttura di due campi:

- ***valore*** dell'elemento
- un ***puntatore*** nodo successivo lista (NULL se ultimo elemento)

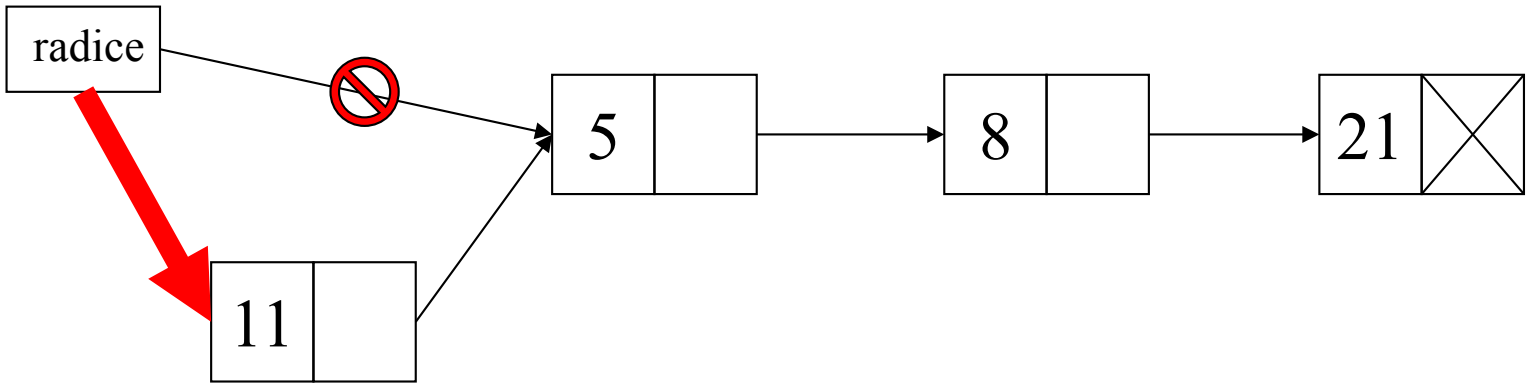



```
typedef struct list_element {
      int value;
      struct list_element *next;} item;
typedef item *list;
list radice = NULL;
```

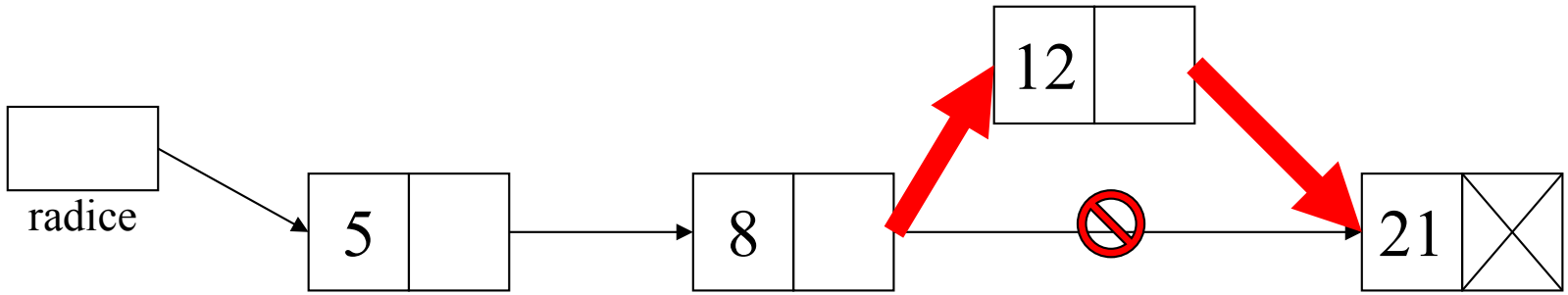
NOTA: ***etichetta list_element*** in dichiarazione struct è ***indispensabile***, altrimenti sarebbe impossibile definire un tipo ricorsivamente

RAPPRESENTAZIONE COLLEGATA

INSERIMENTO (in testa o …)

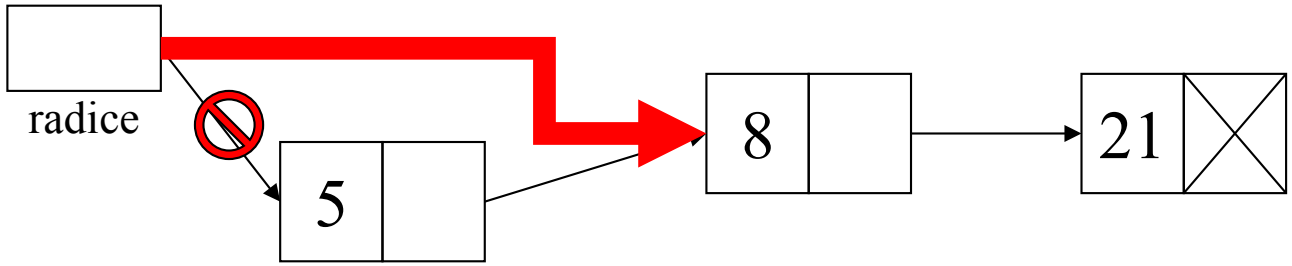


(… in un punto intermedio)

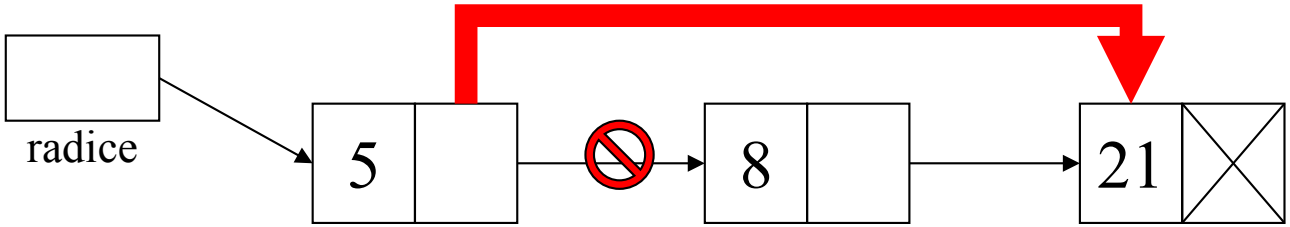


RAPPRESENTAZIONE COLLEGATA

ELIMINAZIONE (dell'elemento in testa o …)



(di un elemento intermedio)



Liste semplici – operazioni su liste

Obiettivi:

- Sperimentare la realizzazione collegata (puntatori a strutture) per la lista semplice
- Definire funzioni (primitive) per la creazione di liste
- Definire funzioni di elaborazione sequenziale su liste, iterative e ricorsive

Liste - RAPPRESENTAZIONE COLLEGATA

IMPLEMENTAZIONE MEDIANTE PUNTATORI

Ciascun nodo della lista è una struttura di due campi:

- ***valore*** dell'elemento
- un ***puntatore*** nodo successivo lista (NULL se ultimo elemento)

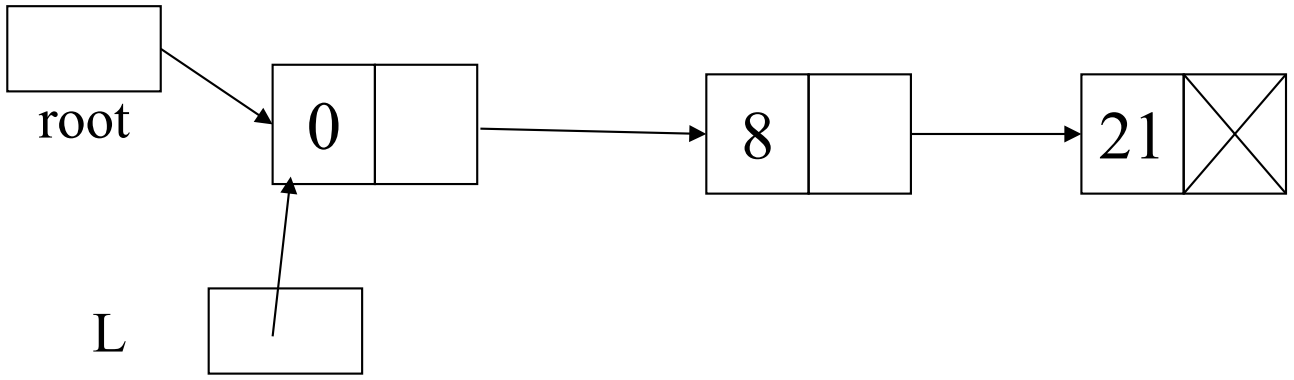
```
typedef struct list_element {
        int value;
        struct list_element *next;} item;
typedef item *list;
```

NOTA: ***etichetta list_element*** in dichiarazione struct è ***indispensabile***, altrimenti sarebbe impossibile definire un tipo ricorsivamente

Creazione di una lista

Il ***main*** da realizzare deve leggere la sequenza di interi e inserire ogni elemento letto in una ***lista***

Per ogni inserimento, **`malloc`** e aggiustamento dei puntatori



root inizialmente NULL

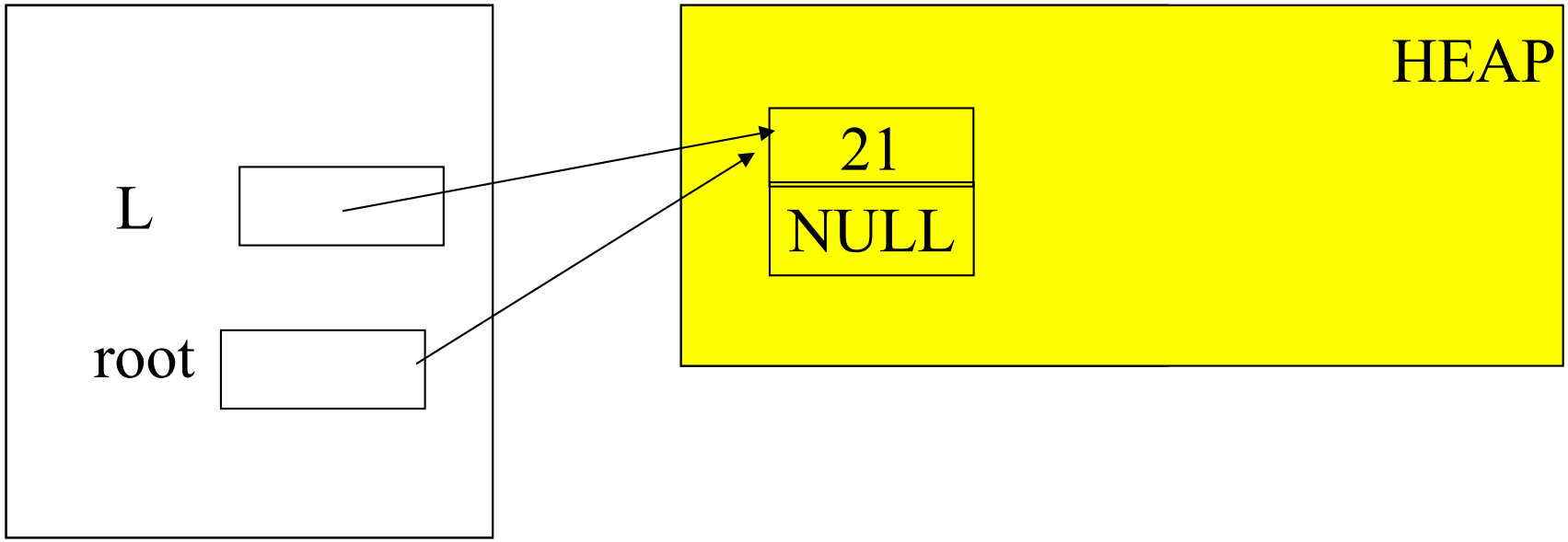
Creazione di una lista semplice

```
#include <stdlib.h>
#include <stdio.h>
typedef struct list_element { int value;
        struct list_element *next; } item;
typedef item *list;

main() { list L; int i;
   list root = NULL;
   do { printf("\nIntrodurre valore: \t");
          scanf("%d", &i);
```

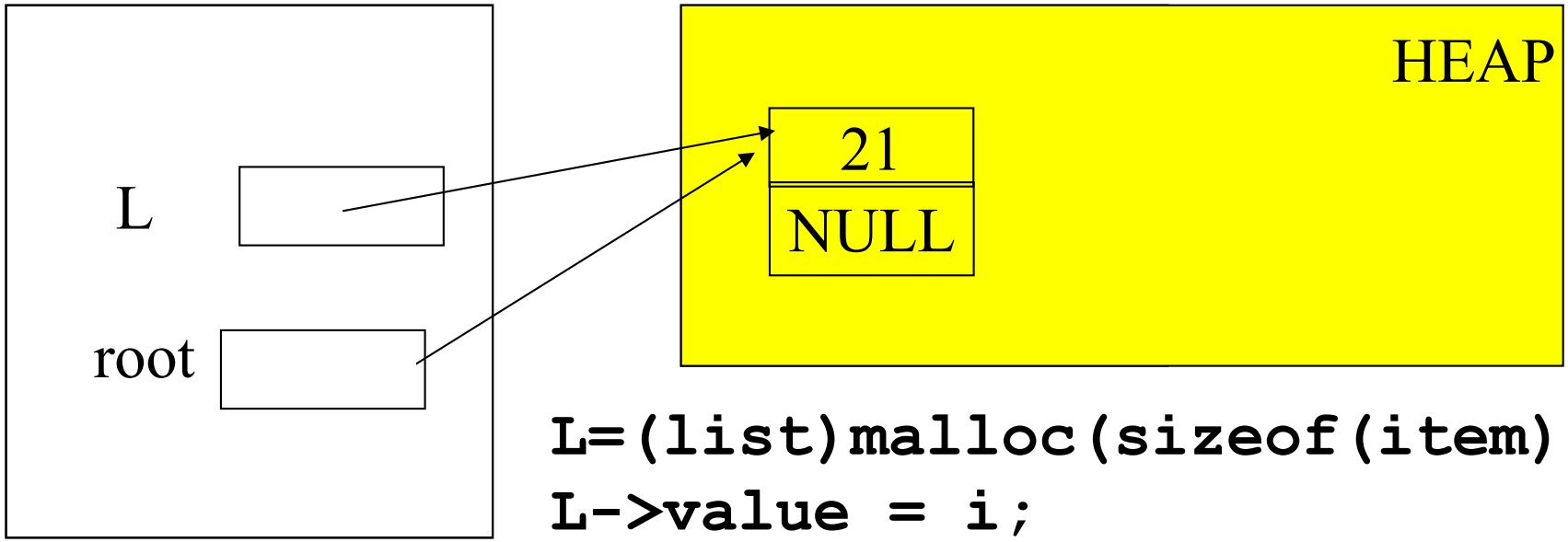
Inserisci **i** nella lista puntata da **root**

```
          } while (i!=0);
}
```

Prima iterazione: leggo 21

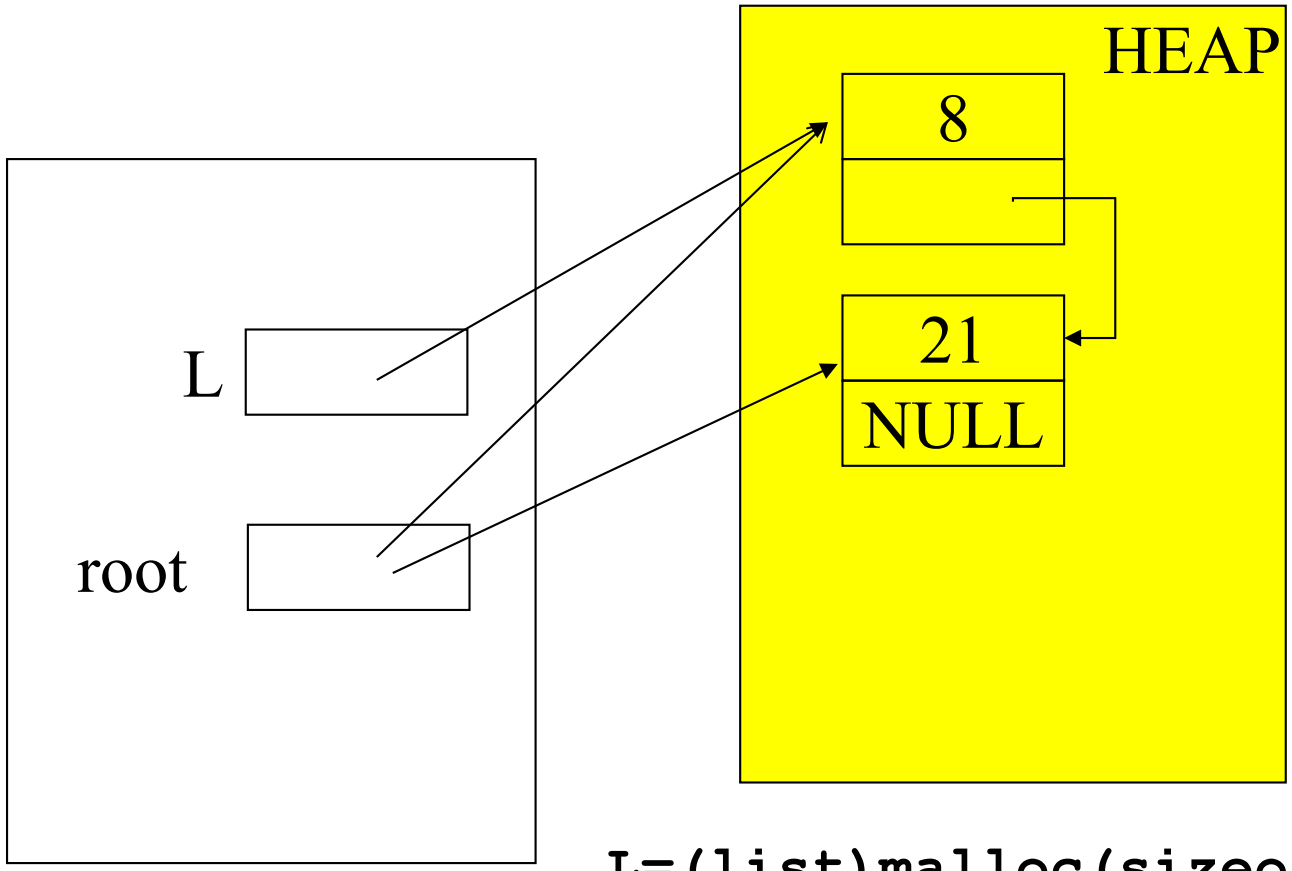


Prima iterazione: leggo 21



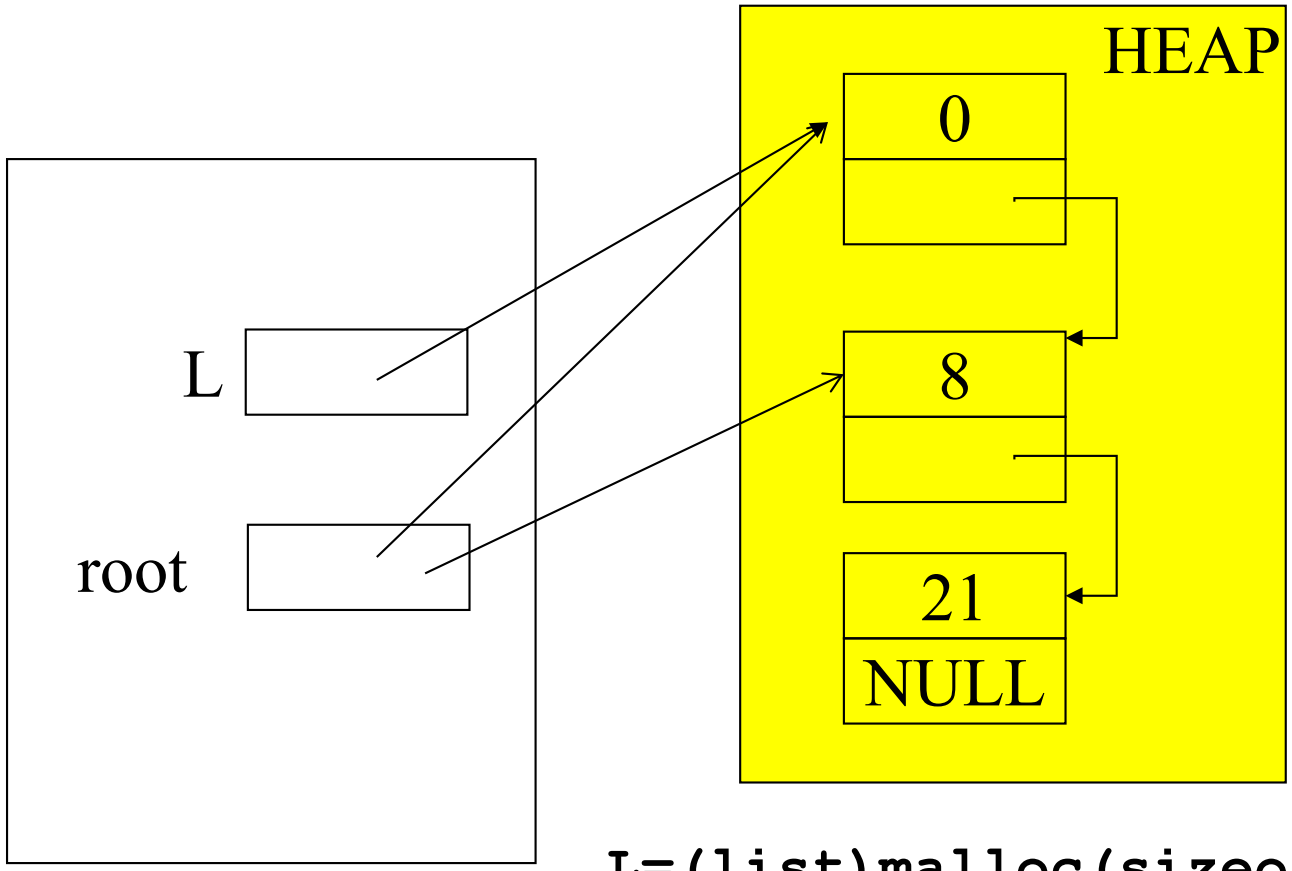

```
L=(list)malloc(sizeof(item));
L->value = i;
L->next = root;
root = L;
```

Seconda iterazione: leggo 8




```
L=(list)malloc(sizeof(item));
L->value = i;
L->next = root;
root = L;
```

Terza iterazione: leggo 0




```
L=(list)malloc(sizeof(item));
L->value = i;
L->next = root;
root = L;
```

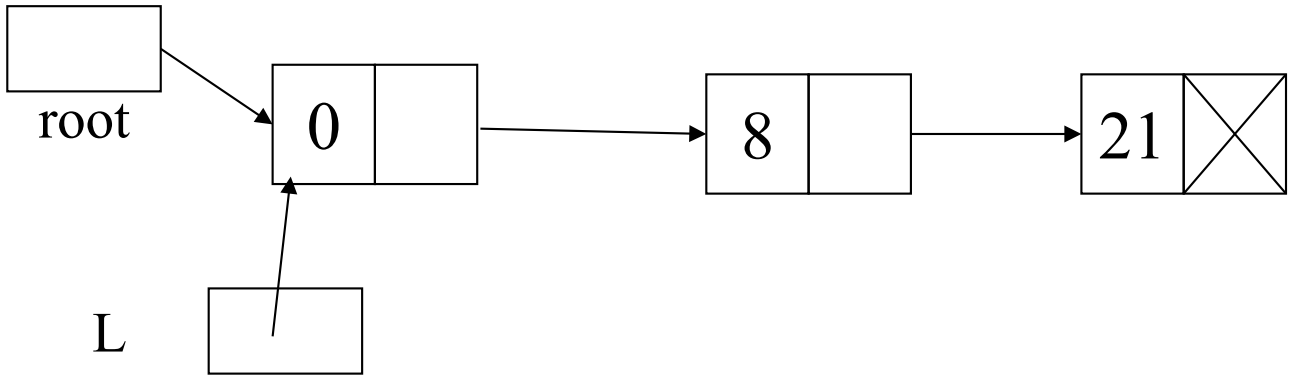
Creazione di una lista semplice

```
#include <stdlib.h>
#include <stdio.h>
typedef struct list_element { int value;
        struct list_element *next; } item;
typedef item *list;

main() { list L; int i;
  list root = NULL;
  do { printf("\nIntrodurre valore: \t");
       scanf("%d", &i);
       L = (list) malloc(sizeof(item));
       L->value = i;
       L->next = root;
       root = L;
     } while (i!=0);
/* stampa della lista */  }
```

Stampa a video di una lista

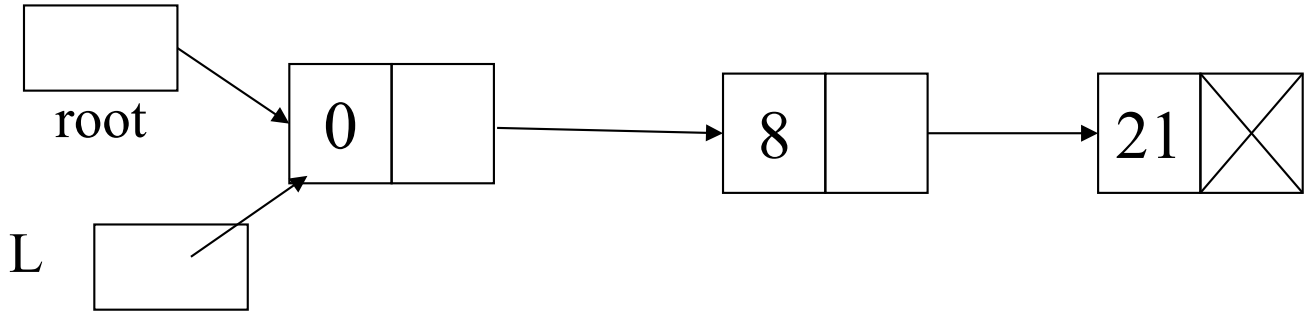
Il ***main*** da realizzare deve leggere la sequenza di interi e inserire ogni elemento letto in una ***lista***

Poi stampiamo la sequenza …
(facciamo insieme anche questo secondo passo)



Non c'è un "indice" da incrementare come per vettori

Stampa di una lista di interi (segue)

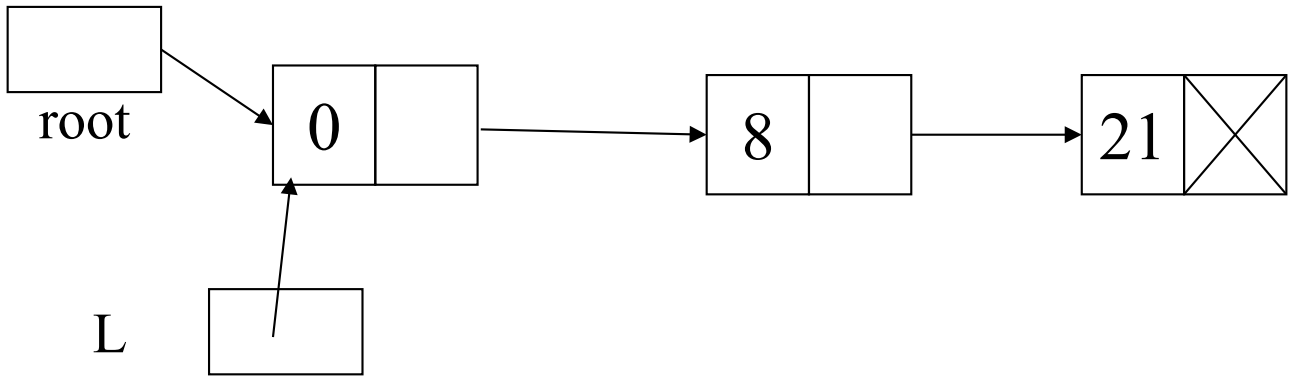



```
i=0;        // se avessimo un vettore …
while (i<N) {
 printf("\nValore : \t%d", V[i]);
 i++; }
}          /* stampa degli elementi di un vettore */
```

```
L=root;
while (L!=NULL) {
 printf("\nValore: \t%d", L->value);
 L = L->next; }
}
```

Stampa di una lista di interi (segue)

```
L = root;

while (L!=NULL) {
 printf("\nValore estratto: \t%d", L->value);
 L = L->next; }
}
```

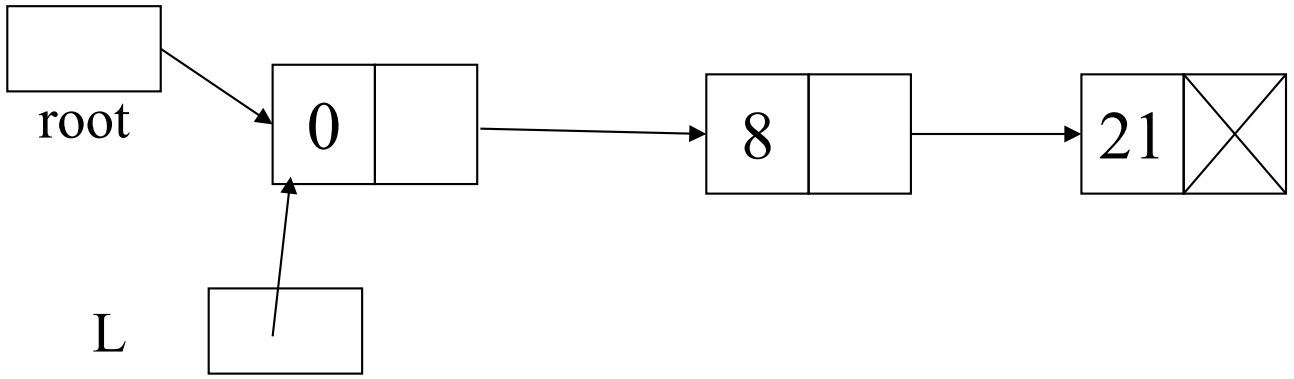


Per "non perdere" il puntatore al primo elemento della lista, la lista va scandita con un puntatore ausiliario (ad esempio L)

FUNZIONI cons E showlist

Prepariamo opportune funzioni di inserimento in testa alla lista e di stampa della lista

```
list cons(int, list);
void showlist (list);
```



Creazione di una lista semplice

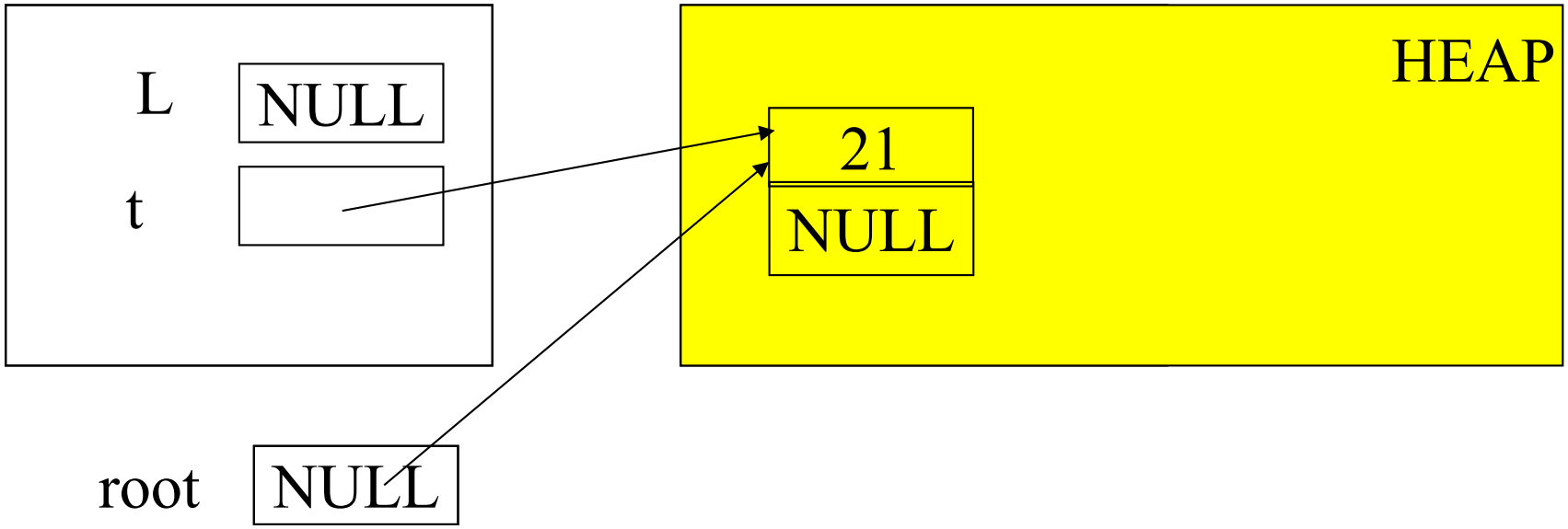
```
#include <stdlib.h>
#include <stdio.h>
typedef struct list_element
    { int value;
      struct list_element *next; } item;
typedef item *list;

main() { list L; int i;
  list root = NULL;
  do { printf("\nIntrodurre valore: \t");
       scanf("%d", &i);
       L = (list) malloc(sizeof(item));
       L->value = i;
       L->next = root;
       root = L;
     } while (i!=0);
/*stampa lista*/ }
```

Creazione di una lista semplice

```
#include <stdlib.h>
#include <stdio.h>
typedef struct list_element
    { int value;
      struct list_element *next; } item;
typedef item *list;

main() { list L; int i;
  list root = NULL;
  do { printf("\nIntrodurre valore: \t");
       scanf("%d", &i);

    root = cons( i, root);

       } while (i!=0);
/*stampa lista*/ }
```

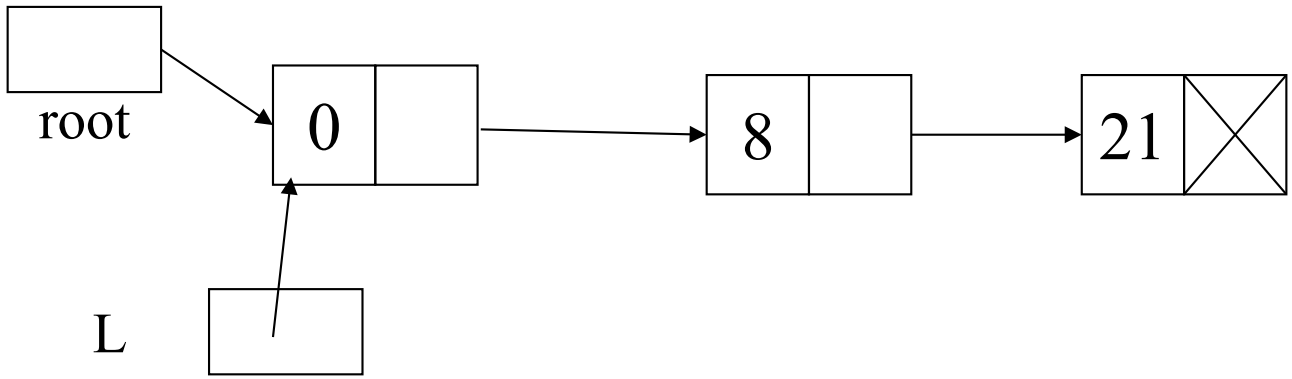
Inserimento in testa: *cons*

```
list cons(int e, list L) {
      list t;
      t = (list) malloc(sizeof(item));
      t->value = e; t->next = L;
      return t; }
```



Stampa di una lista di interi (segue)

```
L = root;

while (L!=NULL) {
 printf("\nValore estratto: \t%d", L->value);
 L = L->next; }
}
```



Per "non perdere" il puntatore al primo elemento della lista, la lista va scandita con un puntatore ausiliario (ad esempio L)

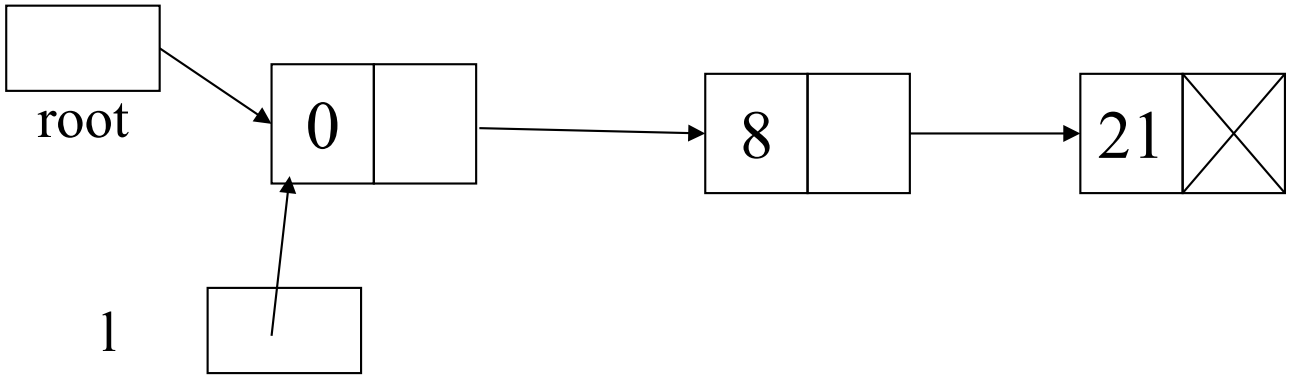
Creazione e stampa di una lista di interi

```
#include <stdio.h>
#include <stdlib.h>
typedef struct list_element { int value;
              struct list_element *next; } item;
typedef item *list;
void showList(list l);

list cons(int e, list l) { list t;
       t = (list) malloc(sizeof(item));
       t->value = e; t->next = l;
       return t; }

main() {
list root = NULL; int i;
do {   printf("\nIntrodurre valore: \t");
       scanf("%d", &i);
       root = cons(i, root);
  } while (i!=0);
showList(root); }
```

Stampa di una lista: `showList`

Stampa di una lista: la lista va ***scandita (sequenzialmente) con un puntatore:***

```
void showList(list l) {
      while ( l!=NULL )
            {   printf("%d", l->value);
                l=l->next;  }
}
```



Iterativa o ricorsiva?

Stampa di una lista: `showListr` ricorsiva

… oppure ***versioni ricorsive***

Stampa di una lista (ricorsiva):

```
void showListr(list l) {
      if( l!=NULL )
            { printf("%d", l->value);
              showListr( l->next ); }
}
```

Rivediamo il programma

Abbiamo visto queste due funzioni per inserire un elemento i in testa ad una lista L:

```
root = cons( i, root);
```

e per stampare il contenuto di una lista L:

```
showList(root);
```

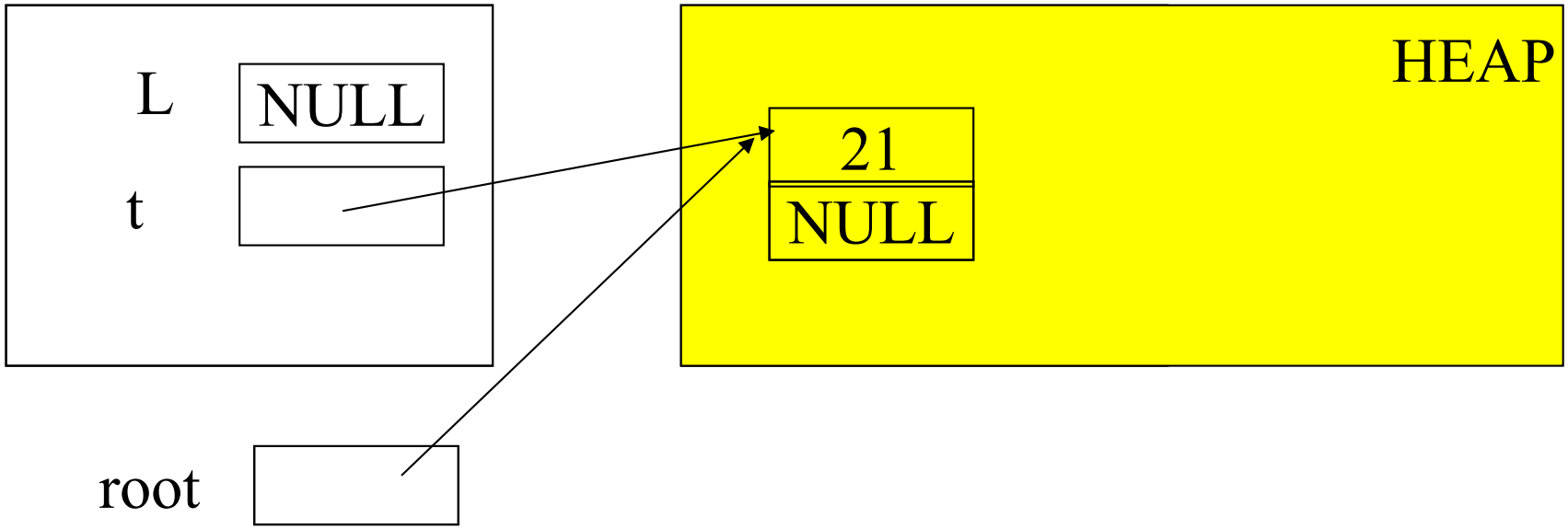
Riscriviamo il ***main*** utilizzando chiamate a ***cons*** e ***showList***

Creazione e stampa di una lista di interi

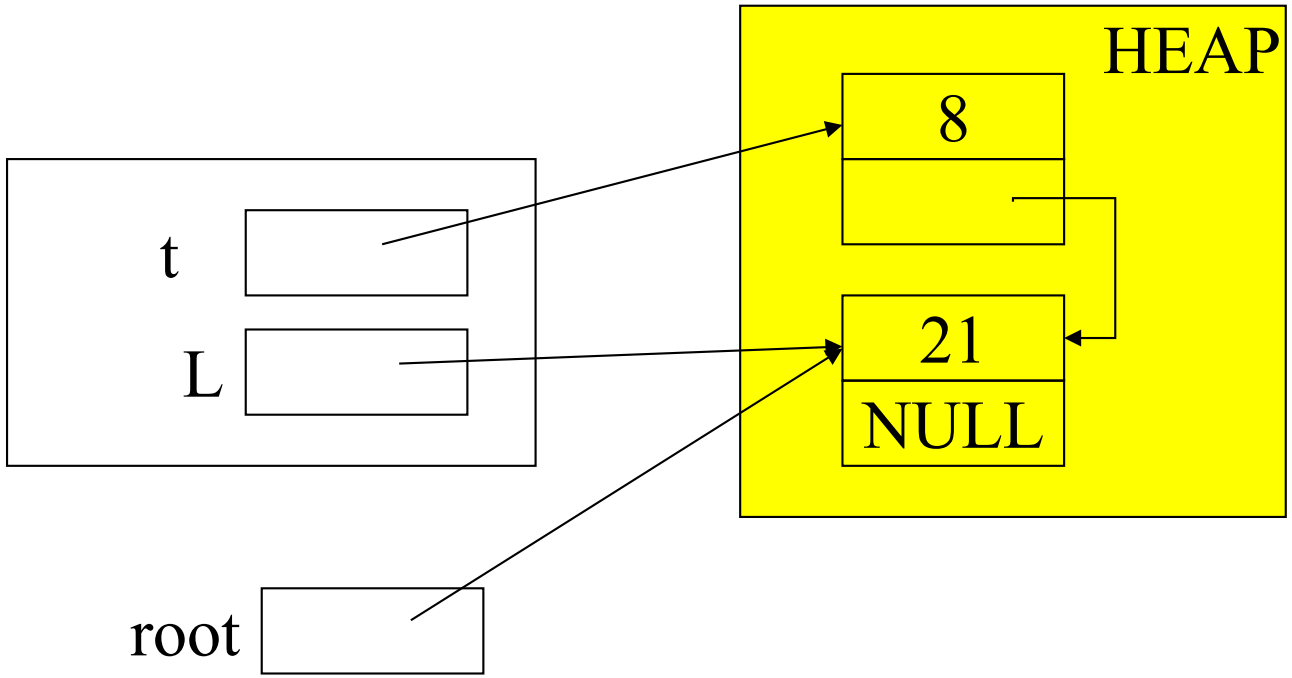
```
#include <stdio.h>
#include <stdlib.h>
typedef struct list_element { int value;
              struct list_element *next; } item;
typedef item *list;
void showList(list l);

list cons(int e, list l) { list t;
       t = (list) malloc(sizeof(item));
       t->value = e; t->next = l;
       return t; }

main() {
list root = NULL; int i;
do {   printf("\nIntrodurre valore: \t");
       scanf("%d", &i);
       root = cons(i, root);
  } while (i!=0);
showList(root); }
```

Prima chiamata: root=cons(i,root)

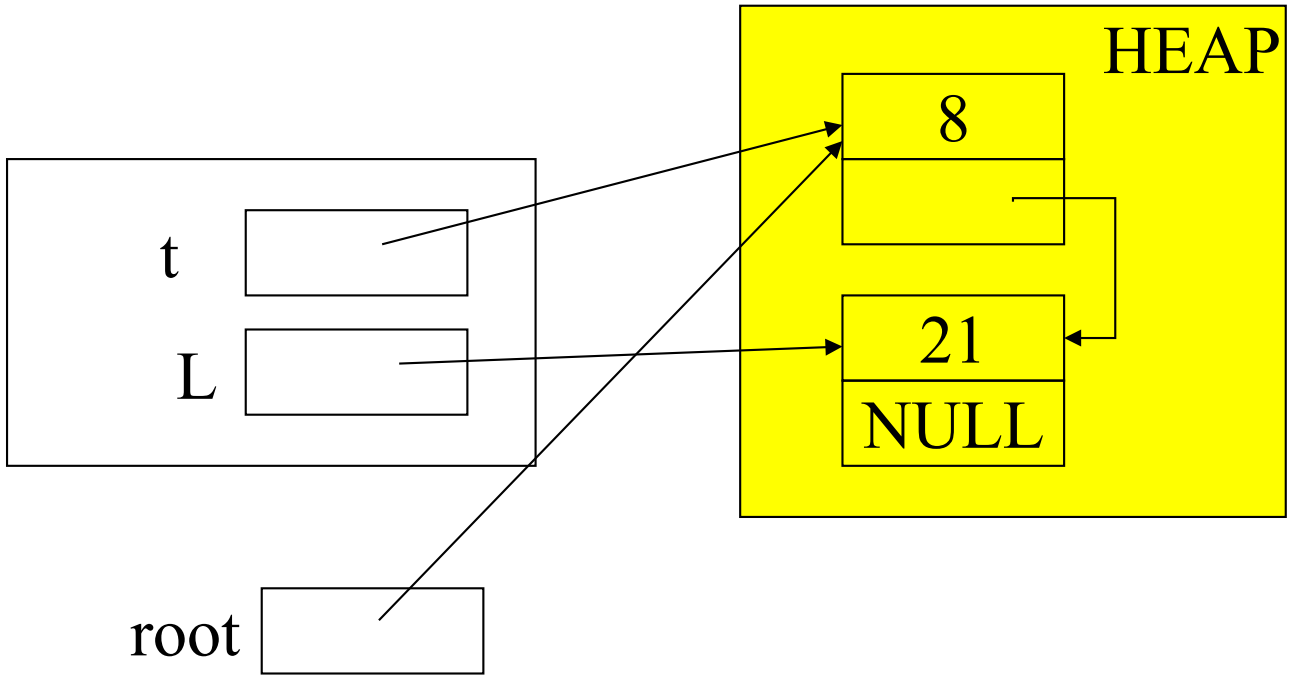
```
list cons(int e, list L) {
        list t;
        t = (list) malloc(sizeof(item));
        t->value = e; t->next = L;
        return t; }
```



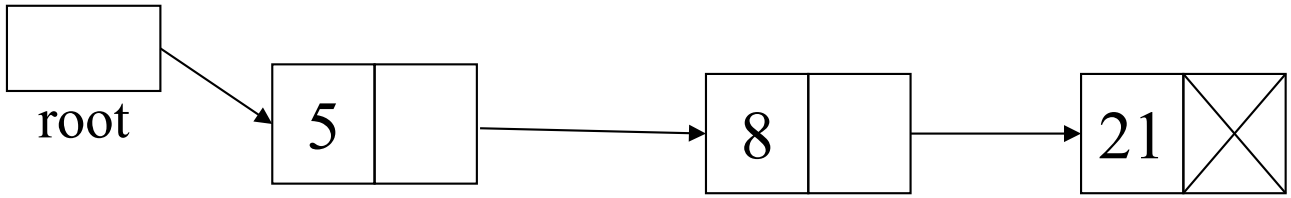
Seconda chiamata: root=cons(i,root)



Seconda chiamata: root=cons(i,root)



Operazioni su liste



Tipiche operazioni:

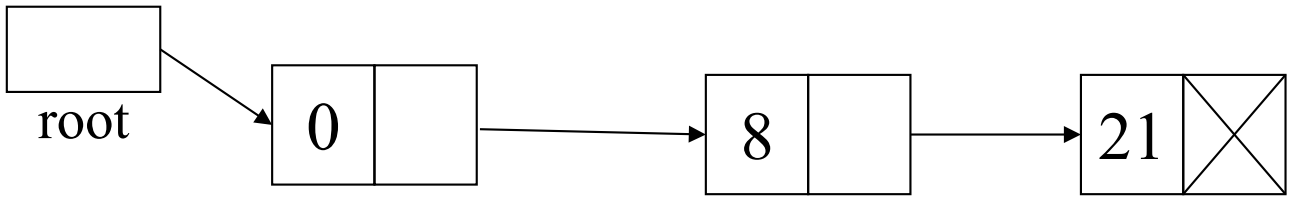
- stampa degli elementi ✅
- ricerca di un elemento
- conteggio degli elementi (lunghezza)
- etc.

Non abbiamo un “indice” da incrementare per scandire la lista, come succedeva per la rappresentazione sequenziale mediante vettori.

La lista va ***scandita (sequenzialmente) con un puntatore,*** *oppure* ***versioni ricorsive***

Ricerca in una lista

```
…
printf(“\nIntrodurre un valore da cercare: \t”);
      scanf(“%d”, &i);
      if (member(i, root)) printf(“Trovato\n”);
       else printf(“Non trovato\n”);

}
```

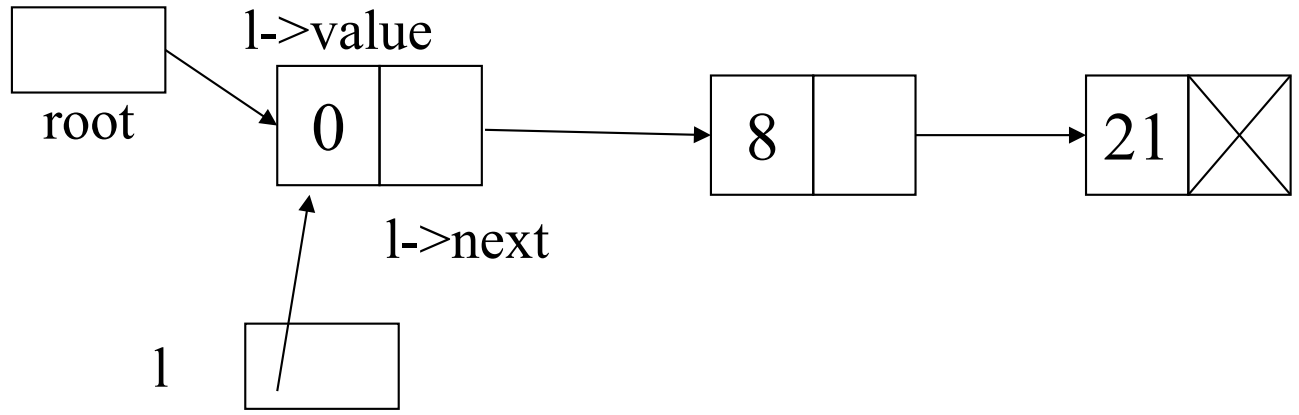


Non abbiamo un “indice” da incrementare per scandire la lista, come succedeva per la rappresentazione sequenziale mediante vettori.

La lista va ***scandita (sequenzialmente) con un puntatore***

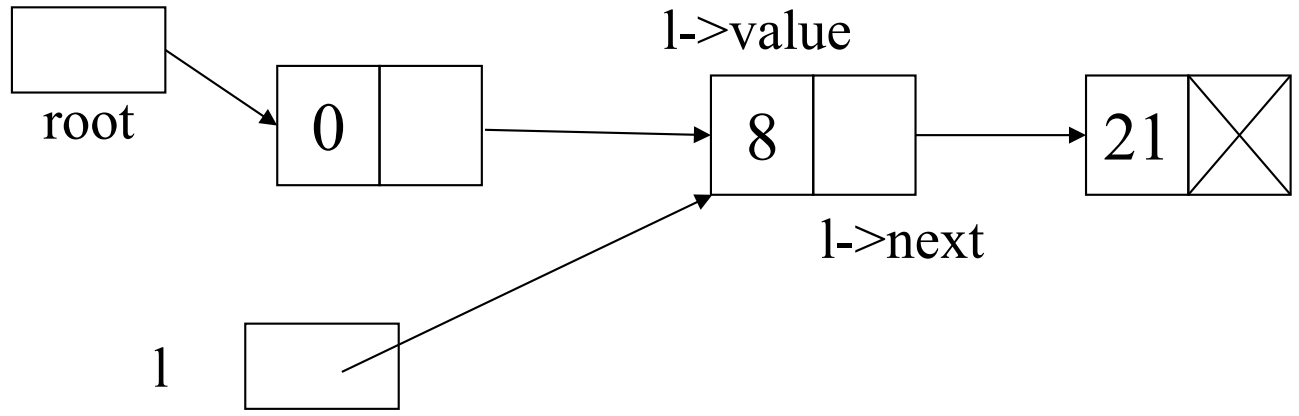
Ricerca in una lista (iterativa)

```
int member(int e, list l) {
      while (l!=NULL)
            if (l->value == e) return 1;
            else l = l->next;
      return 0;
}
```



Ricerca in una lista (iterativa)

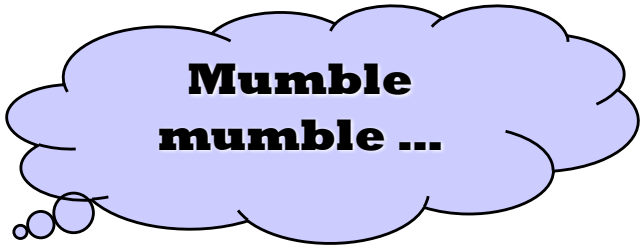
```
int member(int e, list l) {
      int trovato = 0;
      while ((l!=NULL) && !trovato)
            if (l->value == e) trovato = 1;
            else l = l->next;
      return trovato;
}
```



Cerchiamo nella lista

Aggiungiamo al ***main*** che utilizza già chiamate a ***cons*** e ***showList*** anche

la lettura da input di un valore intero



che viene poi cercato in lista con la funzione ***member***

e visualizziamo a video se trovato o meno
if (member(i,root)) printf(“%s”, “trovato”);

Inserimento in testa (*recap*)

```
#include <stdio.h>
#include <stdlib.h>
typedef struct list_element { int value;
                struct list_element *next; } item;
typedef item *list;
void showList(list l);

list cons(int e, list l) { list t;
        t = (list) malloc(sizeof(item));
        t->value = e; t->next = l;
        return t; }

main() {
list root = NULL; int i;
do {    printf("\nIntrodurre valore: \t");
        scanf("%d", &i);
        root = cons(i, root);
  } while (i!=0);
showList(root);
scanf("%d", &i);
If (member(i,root)) printf("%s","trovato"); }
```

ESERCIZIO: ricerca in una lista

```
int member(int e, list l) {
        int trovato = 0;
        while ((l!=NULL) && !trovato)
                if (l->value == e) trovato = 1;
                else l = l->next;
        return trovato;
}
```

È una ***scansione sequenziale,*** **quale complessità?**

- nel caso ***peggiore***, occorre scandire l'intera lista *O(N)*
- nel caso ***migliore***, è il primo elemento *costante*
- nel caso ***medio***, proporzionale a N/2 *O(N)*

Esercizio proposto: progettare e implementare una soluzione ricorsiva (in Laboratorio – **Esercitazione**)

ESERCIZIO 3bis: ricerca in una lista

```
int member(int e, list l) {
        int trovato = 0;
        while ((l!=NULL) && !trovato)
                if (l->value == e) trovato = 1;
                else l = l->next;
        return trovato;
}
```

Soluzione ricorsiva:

ESERCIZIO 3bis: ricerca in una lista

```
int member(int e, list l) {
        int trovato = 0;
        while ((l!=NULL) && !trovato)
                if (l->value == e) trovato = 1;
                else l = l->next;
        return trovato;
}
```

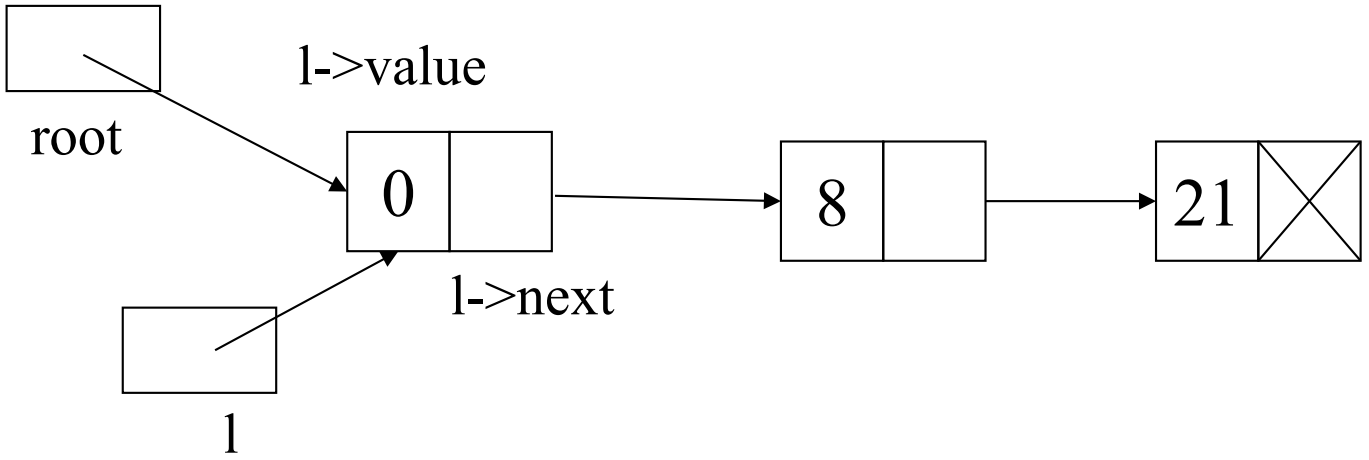
Soluzione ricorsiva:

```
int member_r(int e, list l) {
        if (l!=NULL)
                if (l->value == e) return 1;
                 else  return member_r(e,l->next);
        else return 0;
}
```

Conteggio degli elementi in lista

length(l) =

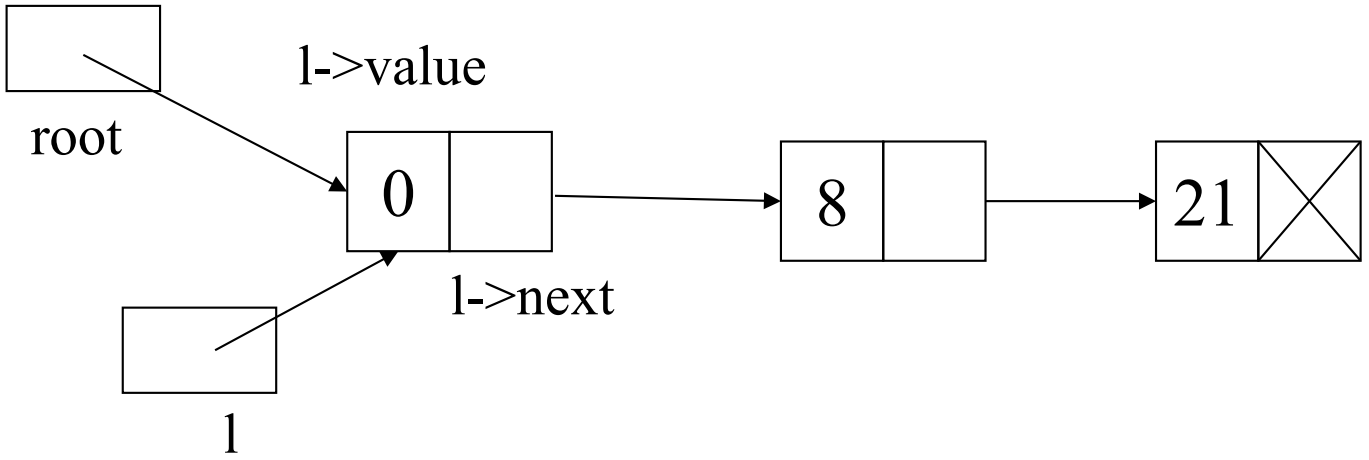
0	se l *è vuota*
1 + length(*resto di* l)	altrimenti



Conteggio degli elementi in lista

length(l) =

0	se l *è vuota*
1 + length(l->next)	altrimenti



Conteggio degli elementi in lista

length(l) = 0 se l è vuota

1 + length(l->next) altrimenti

```
// VERSIONE ITERATIVA
int length(list l) {
        int n = 0;
        while (!(l==NULL)) {
                n++; l = l->next); }
        return n;
}
```

```
// VERSIONE RICORSIVA
int length(list l) {
        if (l==NULL) return 0;
        else return 1 + length(l->next);
}
```

NOTA: ***NON*** è una funzione ***tail ricorsiva***, somma dopo la chiamata ricorsiva

Inserimento di un elemento in una lista

- In testa a una lista ✅

  facile da realizzare

  L'ordine in lista è esattamente inverso rispetto all'ordine di inserimento

- In fondo a una lista

  facile da realizzare?

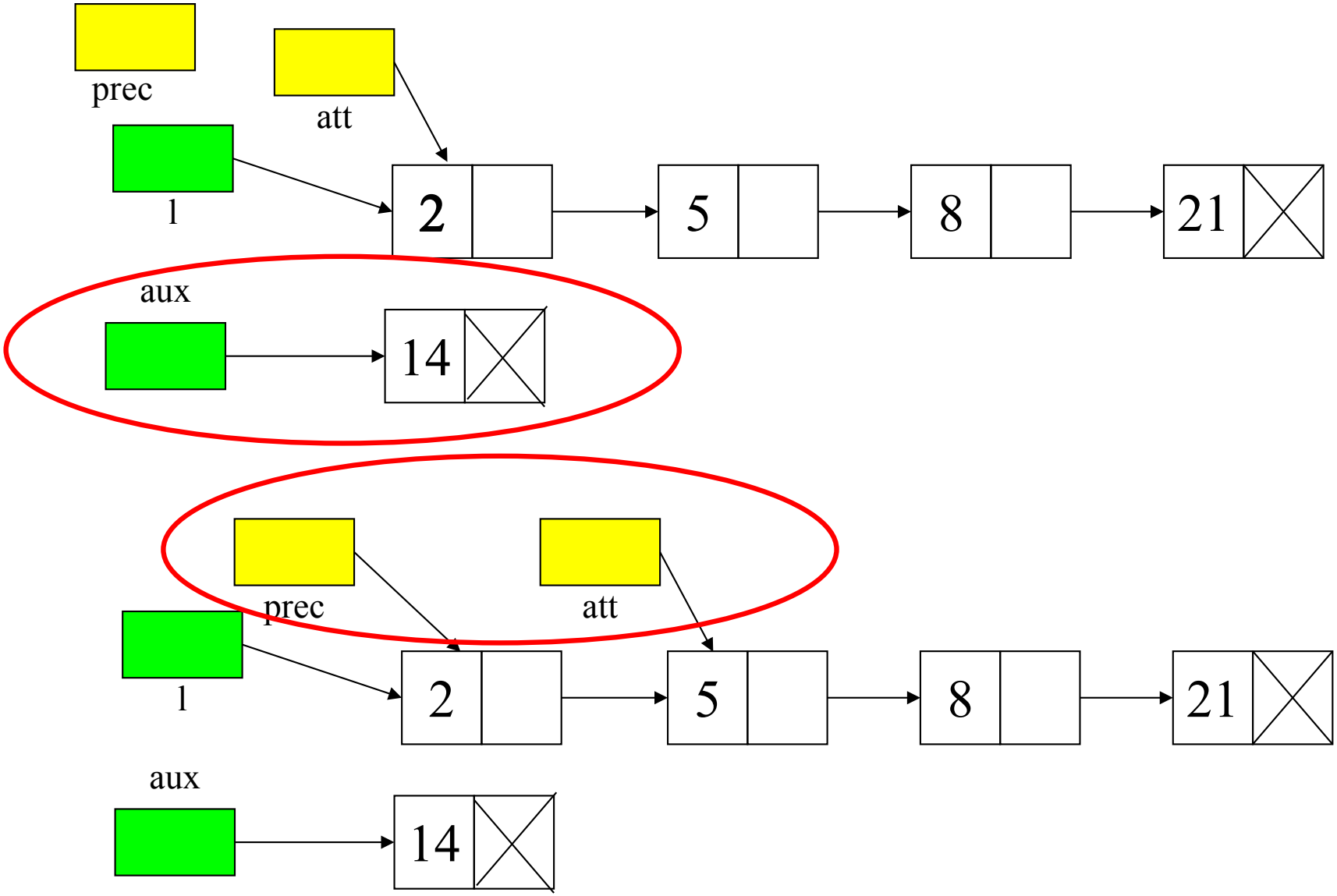
  L'ordine in lista è esattamente identico all'ordine di inserimento

- Inserimento ordinato

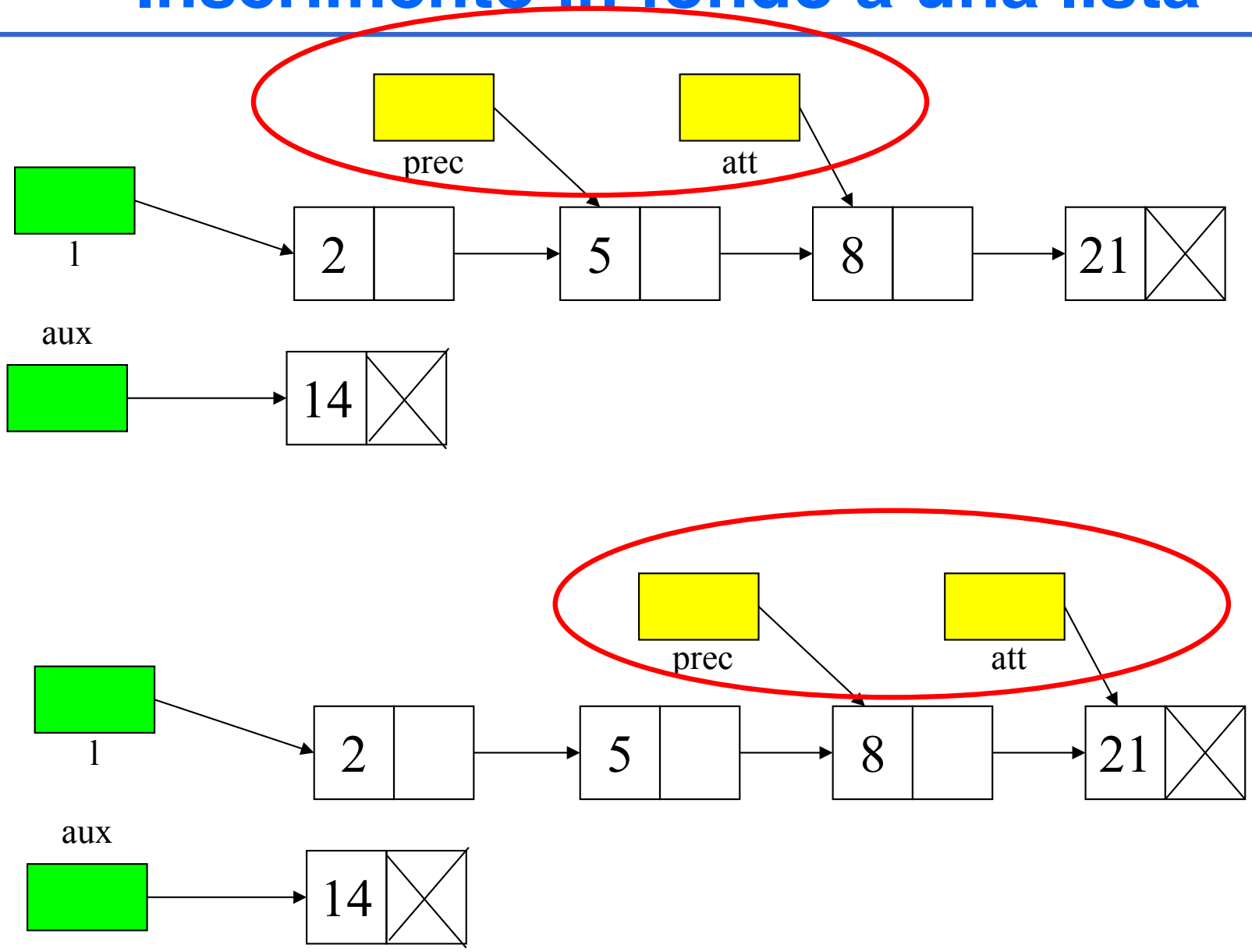
  facile da realizzare?

  L'ordine in lista rispetta la relazione d'ordine tra elementi utilizzata (crescente / decrescente / etc)

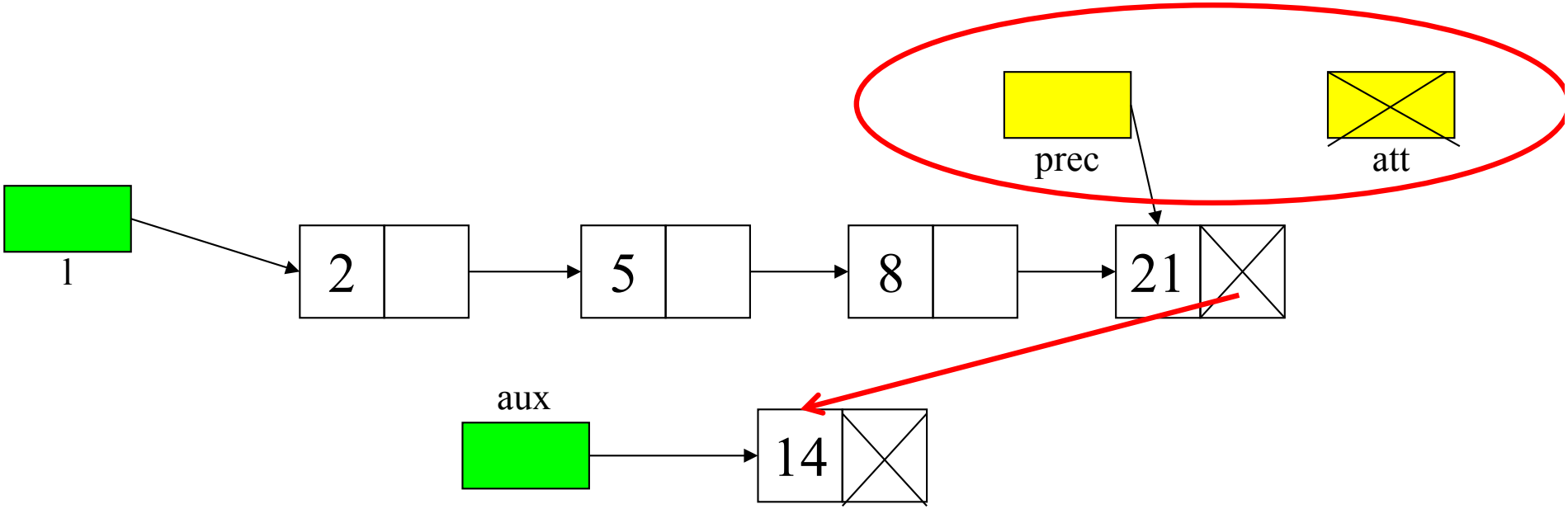
Inserimento in fondo a una lista



Inserimento in fondo a una lista



Inserimento in fondo a una lista



Inserimento in fondo

```
// FUNZIONE CHE INSERISCE IN FONDO - PRIMITIVA ITERATIVA

list  cons_tail(int e, list l) {
  list prec, aux;
  list patt=l;

  aux=(list)malloc(sizeof(item));          // ALLOCA NODO
  aux->value=e;
  aux->next=NULL;
  if (l==NULL) return aux;     // INSERISCE IN LISTA VUOTA
  else
    { while (patt!=NULL)              // NON FINE LISTA
        { prec=patt ;
          patt=patt->next; }

      prec->next=aux;                 // AGGIUNGE IN FONDO
      return l;                       // RESTITUISCE RADICE l
    }
}
```

Creazione di una lista inserendo in fondo

Aggiungiamo al ***main*** che utilizza già chiamate a ***cons*** e ***showList*** anche ***cons_tail***

Lettura di una sequenza da input e costruzione di due liste: root con inserimento in testa e L2 in fondo

Stampa di root e L2 con ***showlist***

Come si presentano le due sequenze visualizzate?

Creazione e stampa di una lista di interi

```
#include <stdio.h>
#include <stdlib.h>
typedef struct list_element { int value;
              struct list_element *next; } item;
typedef item *list;
void showList(list l);
list cons(int e, list l);
list cons_tail(int e, list l);

main() {
list root = NULL; int i; list L2=NULL;
do {   printf("\nIntrodurre valore: \t");
       scanf("%d", &i);
       root = cons(i, root);
       L2=cons_tail(i, L2)
  } while (i!=0);
showList(root);
showList(L2);
 }
```