

memoria
dinamica
e liste

array con limiti variabili:

```
int a, b;
```

```
int A[a][b];
```

```
cout<< sizeof(A); //??
```

```
int a, b;
```

```
cin>>a>>b;
```

```
int A[a][b];
```

```
cout<< sizeof(A); //??
```

e se `-2 >> a` ?

meglio non usarli, ma usare la memoria dinamica

il C++ permette di chiedere da programma al sistema operativo l'allocazione di memoria da usare nel programma:

memoria per qualsiasi tipo, intero, double, enum, struct, array ...

new è la funzione che fa la richiesta, essa restituisce il puntatore alla memoria allocata:

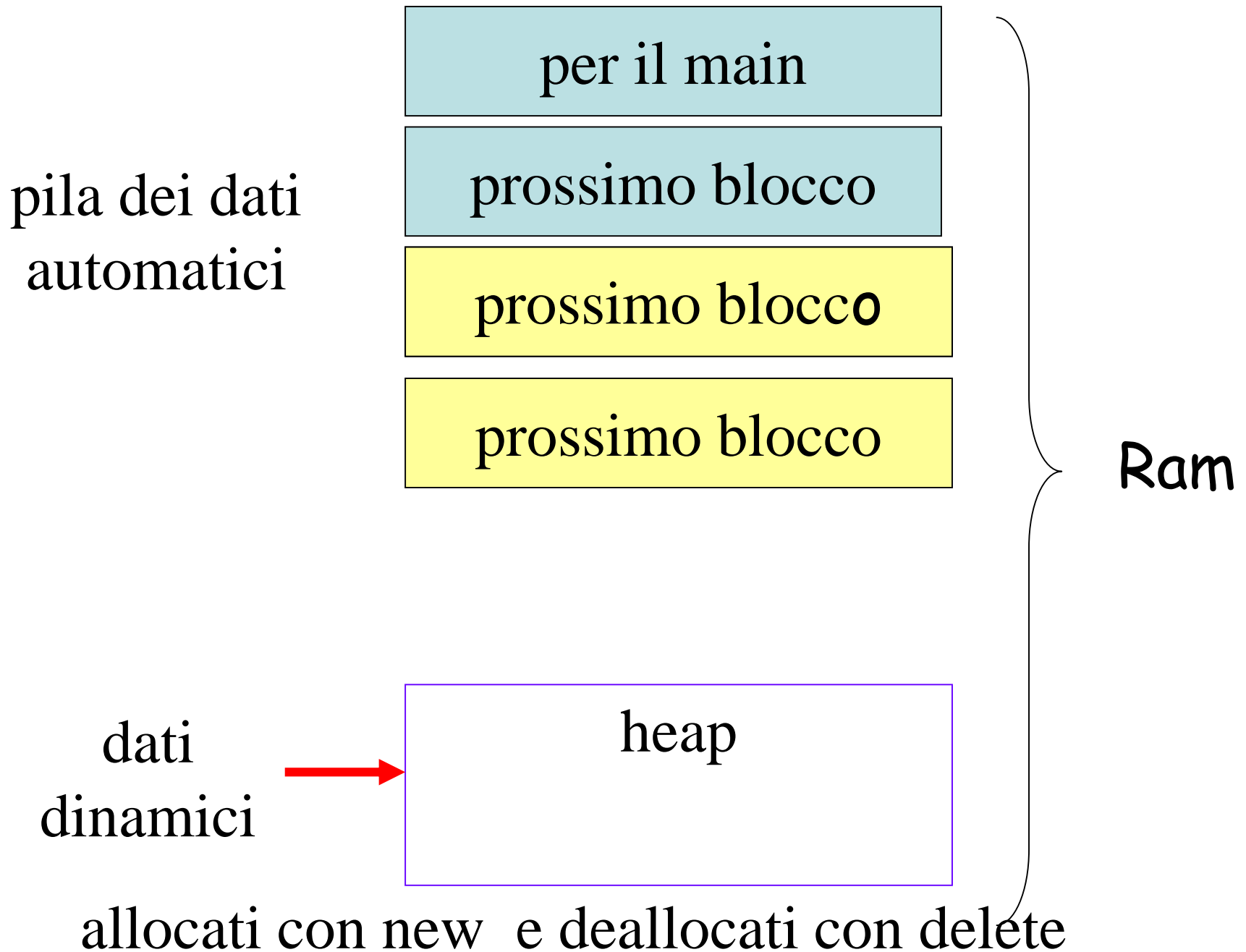
```
int *p=new int;
```

alloca spazio Ram per un intero, p punta a questa locazione Ram

la memoria richiesta con la new viene allocata
sullo HEAP che è diverso dallo stack

la deallocazione deve essere fatta
esplicitamente da programma con la funzione
delete p;

dove p punta all'oggetto da deallocare



allocazione e deallocazione di array;

```
int * p= new int[10];
```

```
delete[] p;
```

anche a più dimensioni:

```
int (*p)[10]=new int [5][10];
```

```
delete[] p;
```

```
int (*p)[8][10]=new int[5][8][10];
```

```
delete[] p;
```

possiamo adattare gli array al bisogno della particolare esecuzione del programma:

supponiamo di avere un programma che legge da cin in un array fino a che trova la sentinella -1 e che deve adattarsi a quanti interi vengono inseriti.

```
main()
{int *p=new int[10], dim=10, i=0;
bool sentinella=false;
while( !sentinella)
{if(i==dim) allunga(p,dim);
cin>>p[i];

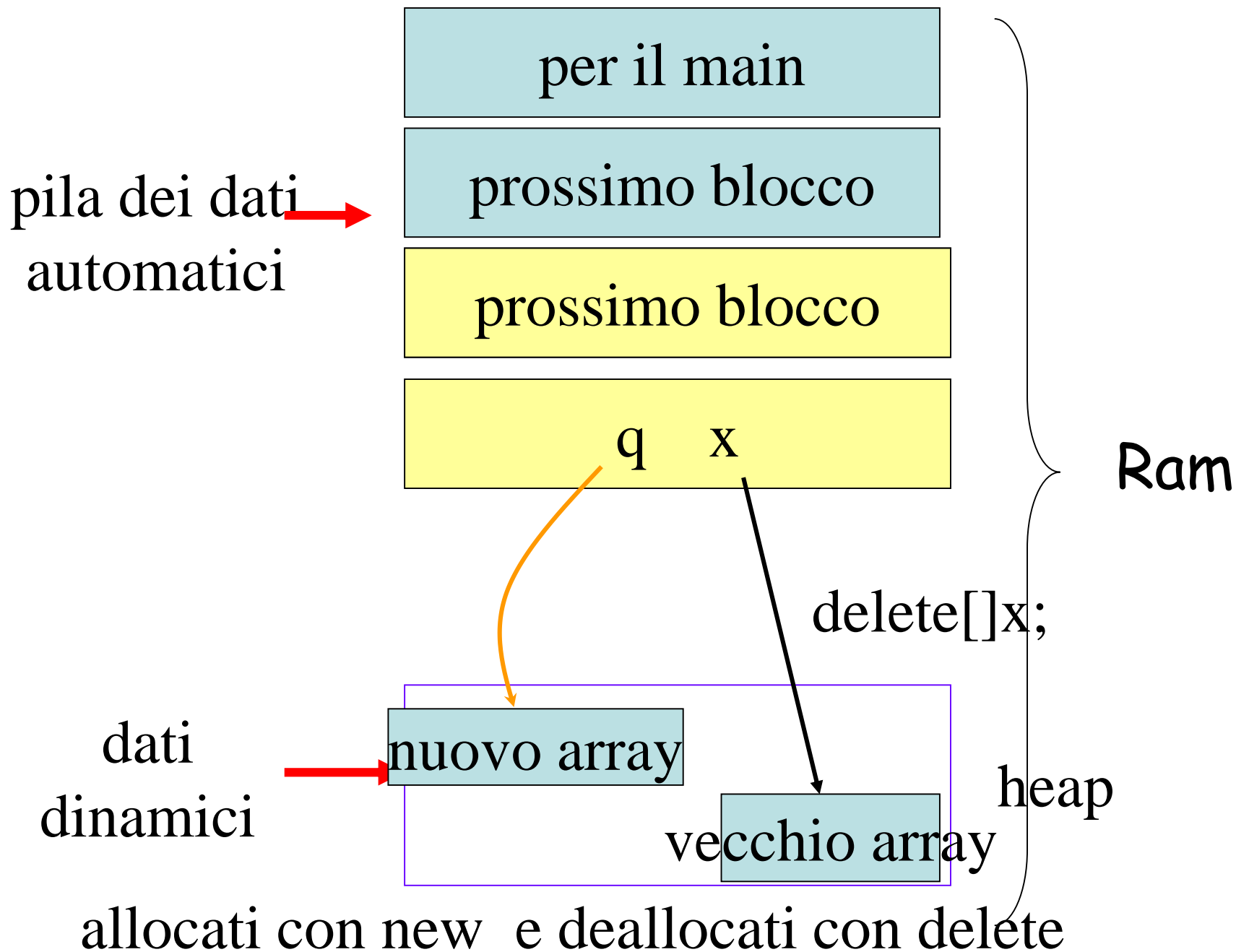
if(p[i]==-1)
    sentinella=true;
else
    i++;
} // i valori letti,

} // p ha dim elementi
```



```
void allunga(int *& x, int & dim)
{
    int * q= new int[dim*2]; // nuovo
    for(int i=0; i<dim; i++)
        q[i]=x[i]; // ricopio il vecchio
    delete [] x; // elimino vecchio
    x=q; // x punta al nuovo array
    dim=dim*2; // dim è nuova dimensione
}
```

notare: q non va deallocata è variabile locale di allunga e quindi viene deallocata automaticamente (sta sulla pila)



con new e delete possiamo costruire strutture
dati capaci di cambiare dinamicamente a
seconda del bisogno

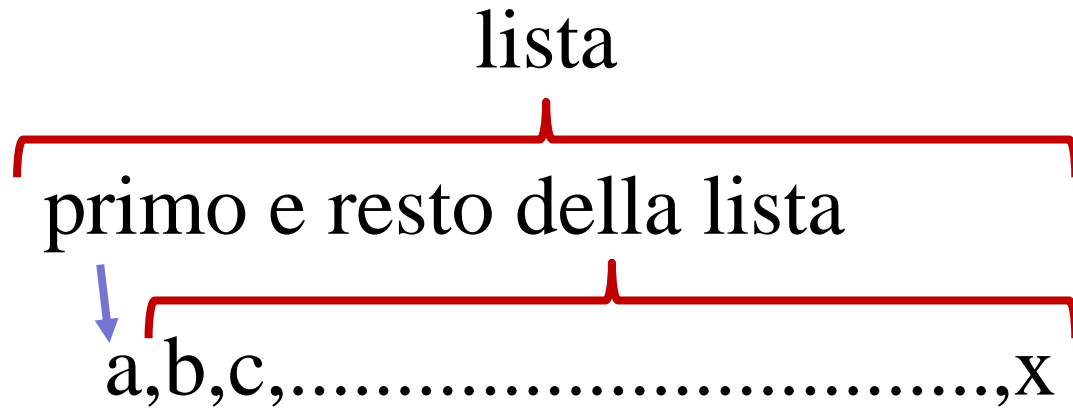
liste e alberi

le liste possono allungarsi e accorciarsi e gli alberi
possono crescere o venire potati

una lista di elementi è una struttura dati ricorsiva,
infatti è definita ricorsivamente con

caso base: una lista vuota

caso ricorsivo: un elemento seguito da una lista di
elementi (detta il resto della lista)

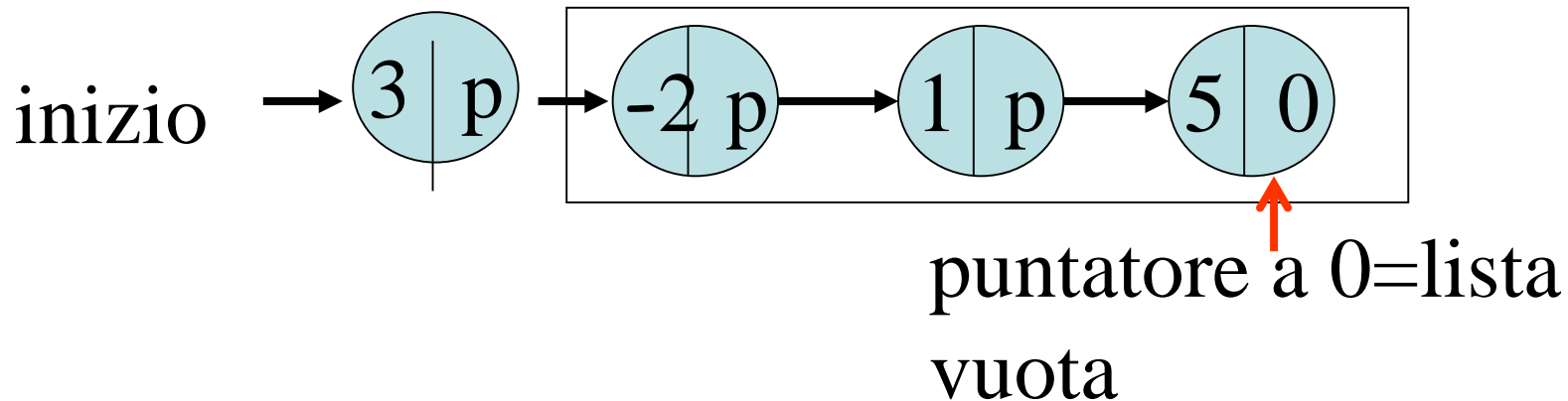
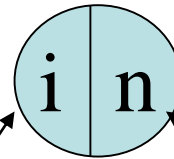


c'è un ordine in una lista non vuota: primo, secondo, ..., ultimo elemento

realizzazione di liste concatenate in C++

ogni nodo della lista ha 2 campi:

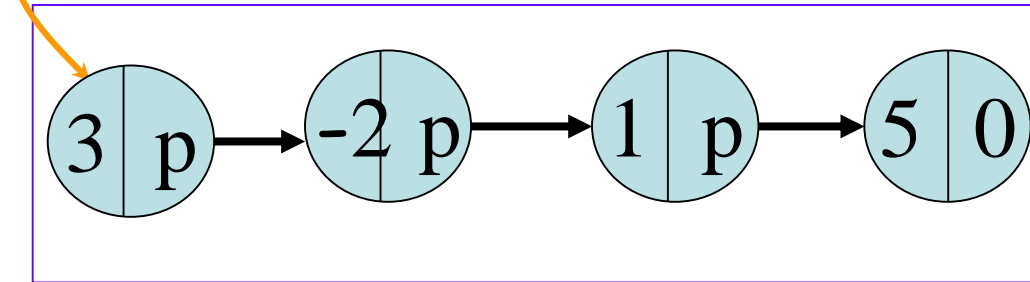
```
struct nodo {int info; nodo* next;};
```



pila dei dati
automatici



heap



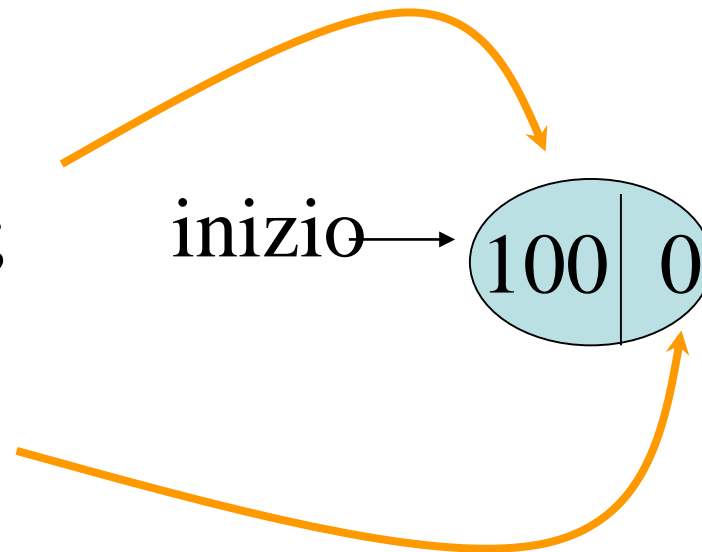
allocando i nodi dinamicamente con new ed eliminandoli con delete possiamo avere liste che crescono (si aggiungono nodi) e diminuiscono (si eliminano nodi) dinamicamente

lista vuota: `nodo * inizio=NULL;`

`inizio=new nodo;`

`(*inizio).info=100;`

`(*inizio).next=0;`



disponibili notazioni diverse:

$(*\text{inizio}).\text{info}=100;$ $\text{inizio} \rightarrow \text{info}=100;$

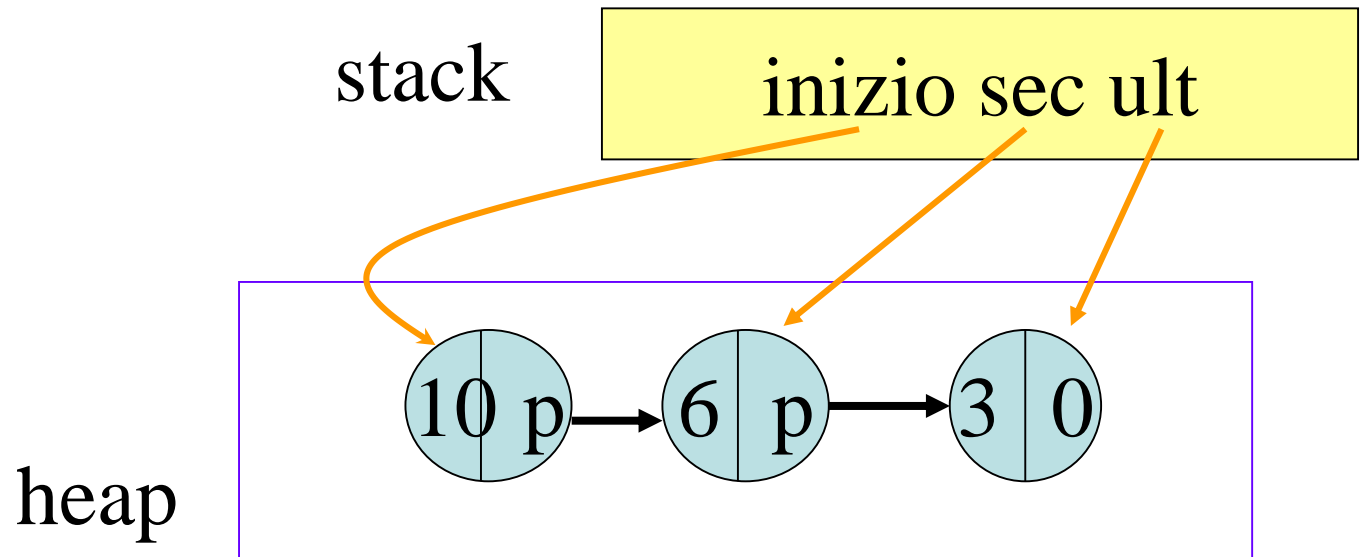
$(*\text{inizio}).\text{next}=0;$ $\text{inizio} \rightarrow \text{next}=0;$

```
struct nodo{int info; nodo* next; nodo(int a=0,  
nodo* b=0){info=a; next=b;}};
```

```
nodo * ult=new nodo(3,0);
```

```
nodo* sec=new nodo(6,ult);
```

```
nodo * inizio=new nodo(10,sec);
```



una lista è un valore di tipo nodo*

una lista si dice corretta quando:

- è 0 (o NULL)

- punta a un nodo il cui campo next è a sua volta una lista corretta

in pratica, è una sequenza possibilmente vuota di nodi, in cui ciascun nodo punta al prossimo fino all'ultimo che ha puntatore 0

stampa di liste concatenate

-stampa in ordine

-in ordine inverso: dall'ultimo al primo

le facciamo ricorsivamente

```
void stampa(nodo *x)
{
    if(x)
    {
        cout<< x->info;

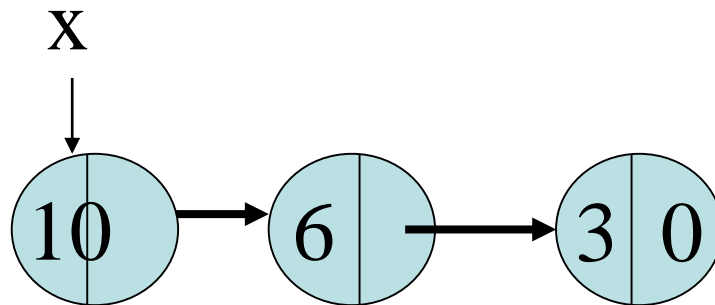
        stampa(x->next);
    }
}
```

ricorsione terminale

si può fare anche col while

```
nodo *x=inizio;  
while(x!=0){  
  cout<< x->info<<endl;  
  x=x->next; }  

```



stampa dal fondo

```
void stampa_rov(nodo *x)
{
    if(x)
    {
        stampa_rov(x->next);
        cout<< x->info;
    }
}
```


la new può fallire !!

potrebbe non esserci memoria Ram sufficiente

```
int * x= new int[1000];
```

```
if(x==NULL) // la new è fallita
```

```
throw(..);
```

la costante predefinita NULL ha valore 0