Iterazione versus ricorsione

1. Introduzione e requisiti del problema
2. Specifica
3. Progetto della soluzione
4. Codifica

Requisiti del problema

Si realizzino in C++ una funzione per la stampa a ritroso dei nodi di una lista ed una per la distruzione dei nodi.
Per entrambe le funzioni si progetti una versione che faccia uso della ricorsione e una che usi solo costrutti iterativi.

In questo esercizio si mettono a confronto iterazione e ricorsione nel realizzare semplici funzioni su liste.
In particolare si mostra la stampa in ordine inverso del valore dei nodi di una lista e la distruzione dei nodi di una lista.
In entrambi i casi la ricorsione offre uno strumento semplice ed elegante per risolvere il problema, mentre l'uso dell'iterazione è meno conciso.

Per questo esercizio sono necessarie conoscenze sulla ricorsione e sull'iterazione.

Casi di test

caso 1: inserimento di una lista non vuota

Input: 3 4 56 7 8 9 4 7 8
Output: 8 7 4 9 8 7 56 4 3

caso 2: lista vuota

Input: lista vuota
Output: lista vuota

- Confronto tra
  iterazione e ricorsione
- Utilizzo dell'esempio delle
  liste dinamiche

- L'iterazione utilizza i cicli (**do**, **for**) per leggere tutta la struttura dati

- Nel caso di strutture dinamiche saranno presenti istruzioni del tipo:

```
puntatoreAelemento=puntatoreAelemento->successivo
```

- I metodi con ricorsione sono composti da due parti:
  - la ***condizione di uscita***
  - il ***passo di ricorsione***

Algoritmo di base
Lettura della lista dalla testa alla coda
Stampa del contenuto dell'elemento
"Risalita" della coda verso la testa della lista

Algoritmo di base

Lettura della lista dalla testa alla coda

Stampa del contenuto dell'elemento

"Risalita" della coda verso la testa della lista

Algoritmo di base
Lettura della lista dalla testa alla coda
Stampa del contenuto dell'elemento
"Risalita" della coda verso la testa della lista

Algoritmo di base
Lettura della lista dalla testa alla coda
Stampa del contenuto dell'elemento
"Risalita" della coda verso la testa della lista

Algoritmo di base
Lettura della lista dalla testa alla coda
Stampa del contenuto dell'elemento
"Risalita" della coda verso la testa della lista

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

Algoritmo di base
Lettura della lista dalla testa alla coda
Stampa del contenuto dell'elemento
"Risalita" della coda verso la testa della lista

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

Algoritmo di base
Lettura della lista dalla testa alla coda
Stampa del contenuto dell'elemento
"Risalita" della coda verso la testa della lista

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

Algoritmo di base
Lettura della lista dalla testa alla coda
Stampa del contenuto dell'elemento
"Risalita" della coda verso la testa della lista

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

Soluzione usando l'iterazione

```
void stampaIterativa(nodo *testa);
```

La struttura dinamica ha un solo puntatore in avanti; dunque non è possibile risalire la lista in maniera diretta.
E' necessario definire un nuovo algoritmo.

Soluzione usando l'iterazione

Se avessimo avuto un array `int lista[MAXLUNG]` sarebbe bastato il seguente ciclo for:

```
for(int i=(MAXLUNG-1);i>=0;i--)
  {
     cout << lista[i];
  }
```

La lista che viene passata alla funzione conterrà un numero noto (o comunque calcolabile) di istanze di termini

possiamo calcolare il numero degli elementi della lista

Soluzione usando l'iterazione

- Uso di una versione modificata del

  ciclo for

- In una lista dinamica non si possono usare *indici*: si può scandire la lista fino all'i-esimo elemento

- Appena stampato un valore si deve ritornare alla testa della lista per la prossima "discesa"

Soluzione usando l'iterazione

2 cicli

```
Ciclo2 (calcolo il valore dell'indice di discesa id)
  {
    Ciclo1(scendo di id volte nella lista)
    Riporto il valore corrente a testa
  }
```

Soluzione usando l'iterazione

2 cicli

```
Ciclo2 (calcolo il valore dell'indice di discesa id)
  {
    Ciclo1(scendo di id volte nella lista)
    Riporto il valore corrente a testa
  }
```

Soluzione usando l'iterazione

2 cicli

```
Ciclo2 (calcolo il valore dell'indice di discesa id)
  {
    Ciclo1(scendo di id volte nella lista)
    Riporto il valore corrente a testa
  }
```

Soluzione usando l'iterazione

2 cicli

```
Ciclo2 (calcolo il valore dell'indice di discesa id)
  {
    Ciclo1(scendo di id volte nella lista)
    Riporto il valore corrente a testa
  }
```

Ciclo1 = for (...)

Soluzione usando l'iterazione

2 cicli

```
Ciclo2 (calcolo il valore del'indice di discesa id)
  {
    Ciclo1(scendo di id volte nella lista)
    Riporto il valore corrente a testa
  }
```

Ciclo1 = for (...)

Ciclo2 = do {...}while () oppure for(...)

Il calcolo del numero degli elementi della lista si ottiene con un terzo ciclo prima del doppio ciclo.

Soluzione usando l'iterazione

Pseudocodice

```
For (fino a che non finisco la lista)
      Incremento di uno il numero degli elementi
Se il numero degli incrementi è zero
      Stampo il messaggio di "lista vuota"
Altrimenti
       Calcolo numero di iterazioni
       Do{
        For (i=0...numero di iterazioni)
          {
            scendo fino al numero di iterazioni stabilite
          }
        stampo il valore corrente
        porto il valore corrente = testa
        decremento il numero delle iterazioni
        }(while numero di iterazioni >=0)
```

Soluzione usando l'iterazione

Pseudocodice

```
For (fino a che non finisco la lista)
      Incremento di uno il numero degli elementi
Se il numero degli incrementi è zero
      Stampo il messaggio di "lista vuota"
Altrimenti
       Calcolo numero di iterazioni
       Do{
        For (i=0...numero di iterazioni)
          {
            scendo fino al numero di iterazioni stabilite
          }
        stampo il valore corrente
        porto il valore corrente = testa
        decremento il numero delle iterazioni
        }(while numero di iterazioni >=0)
```

Soluzione usando l'iterazione

Pseudocodice

```
For (fino a che non finisco la lista)
      Incremento di uno il numero degli elementi
Se il numero degli incrementi è zero
      Stampo il messaggio di "lista vuota"
Altrimenti
       Calcolo numero di iterazioni
       Do{
        For (i=0...numero di iterazioni)
          {
            scendo fino al numero di iterazioni stabilite
          }
        stampo il valore corrente
        porto il valore corrente = testa
        decremento il numero delle iterazioni
        }(while numero di iterazioni >=0)
```

Soluzione usando l'iterazione

Pseudocodice

```
For (fino a che non finisco la lista)
      Incremento di uno il numero degli elementi
Se il numero degli incrementi è zero
      Stampo il messaggio di "lista vuota"
Altrimenti
       Calcolo numero di iterazioni
       Do{
        For (i=0...numero di iterazioni)
          {
            scendo fino al numero di iterazioni stabilite
          }
        stampo il valore corrente
        porto il valore corrente = testa
        decremento il numero delle iterazioni
        }(while numero di iterazioni >=0)
```

Soluzione usando l'iterazione

Pseudocodice

```
For (fino a che non finisco la lista)
      Incremento di uno il numero degli elementi
Se il numero degli incrementi è zero
      Stampo il messaggio di "lista vuota"
Altrimenti
       Calcolo numero di iterazioni
       Do{
        For (i=0...numero di iterazioni)
          {
            scendo fino al numero di iterazioni stabilite
          }
        stampo il valore corrente
        porto il valore corrente = testa
        decremento il numero delle iterazioni
        }(while numero di iterazioni >=0)
```

Soluzione usando l'iterazione

Pseudocodice

```
For (fino a che non finisco la lista)
      Incremento di uno il numero degli elementi
Se il numero degli incrementi è zero
      Stampo il messaggio di "lista vuota"
Altrimenti
       Calcolo numero di iterazioni
       Do{
        For (i=0...numero di iterazioni)
          {
            scendo fino al numero di iterazioni stabilite
          }
        stampo il valore corrente
        porto il valore corrente = testa
        decremento il numero delle iterazioni
        }(while numero di iterazioni >=0)
```

Soluzione usando l'iterazione

Pseudocodice

```
For (fino a che non finisco la lista)
      Incremento di uno il numero degli elementi
Se il numero degli incrementi è zero
      Stampo il messaggio di "lista vuota"
Altrimenti
       Calcolo numero di iterazioni
       Do{
        For (i=0...numero di iterazioni)
          {
            scendo fino al numero di iterazioni stabilite
          }
        stampo il valore corrente
        porto il valore corrente = testa
        decremento il numero delle iterazioni
        }(while numero di iterazioni >=0)
```

Soluzione usando l'iterazione

Pseudocodice

```
For (fino a che non finisco la lista)
      Incremento di uno il numero degli elementi
Se il numero degli incrementi è zero
      Stampo il messaggio di "lista vuota"
Altrimenti
       Calcolo numero di iterazioni
       Do{
        For (i=0...numero di iterazioni)
          {
            scendo fino al numero di iterazioni stabilite
          }
        stampo il valore corrente
        porto il valore corrente = testa
        decremento il numero delle iterazioni
        }(while numero di iterazioni >=0)
```

Soluzione usando l'iterazione

```
void distruggilterativa(nodo *testa);
```

Scelta di progetto:

distruggere la lista partendo dalla testa anziché al contrario

Algoritmo di deallocazione:

```
while (lista vuota)
  {
    tmp = elemento corrente
    elemento corrente = elemento corrente->next
    delete tmp
  }
```

Soluzione usando la ricorsione

```
Funzione stampaRicorsiva (nodo *testa)
Definizione condizione di uscita  → testa==0
Passo di ricorsione               → stampaRicorsiva(testa->next);
```

dopo il passo di ricorsione verrà scritto il codice per stampare i dati.

Soluzione usando la ricorsione

Pseudocodice

```
if testa==0
  stampa "lista vuota"
else
  stampaRicorsiva(testa->next)
  stampa valore corrente di testa->info
```

Data la lista: 7 6 0

Soluzione usando la ricorsione

Pseudocodice

```
if testa==0
  stampa "lista vuota"
else
  stampaRicorsiva(testa->next)
  stampa valore corrente di testa->info
```

Data la lista: 7 6 0

Soluzione usando la ricorsione

Pseudocodice

```
if testa==0
  stampa "lista vuota"
else
  stampaRicorsiva(testa->next)
  stampa valore corrente di testa->info
```

Data la lista: 7 6 0

Soluzione usando la ricorsione

Pseudocodice

```
if testa==0
  stampa "lista vuota"
else
  stampaRicorsiva(testa->next)
  stampa valore corrente di testa->info
```

Data la lista: 7 6 0

Soluzione usando la ricorsione

Pseudocodice

```
if testa==0
  stampa "lista vuota"
else
  stampaRicorsiva(testa->next)
  stampa valore corrente di testa->info
```

```
lista vuota
6
7
```

Soluzione usando la ricorsione

Modificare la condizione di uscita

uscita → testa->next==0

Problemi nel caso in cui testa=0

Soluzione usando la ricorsione

Pseudocodice

```
if testa==0
  stampa "lista vuota"
else if (testa->next!=0)
  {
    stampaRicorsiva(testa->next)
    stampa valore corrente di testa->info
  }
```

```
void distruggiRicorsiva(nodo *testa);
```

```
Condizione di uscita temp==0;
Passo di ricorsione distruggiRicorsiva(temp->next)
```

Iterazione versus ricorsione

La ricorsione consente di scrivere il codice in modo più semplice e veloce dell'iterazione.

Esempio

lista di 30 elementi
Usando l'algoritmo ricorsivo avremo 30 operazioni del tipo:

```
elemento=elemento->next;
cout << elemento->info;
```

Iterazione versus ricorsione

Usando l'algoritmo iterativo avremo 30 assegnamenti per sapere quanto è lunga la lista

$$\sum_{1}^{30} i$$ + 30 assegnamenti e 30 stampe

La ricorsione è sempre meglio dell'iterazione?
Non sempre

Le funzioni ricorsive tendono a occupare molto spazio (a causa della necessità di salvare tutte le informazioni delle funzioni ogni volta che si effettua una chiamata ricorsiva).