```c
/*
 * Corso di Fondamenti di Informatica
 * Esercizio:
 * Si scriva un sottoprogramma che riceve in ingresso la testa di una lista
 * dinamica contenente caratteri e modifica il contenuto della lista invertendo
 * l'ordine degli elementi rispetto all'ordine originale.
 */

#include <stdlib.h>
#include <stdio.h>

typedef struct EL
{
  char        Dato;
  struct EL*  pProssimo;

} ElemLista;
/* tipo di dato degli elementi della lista */


ElemLista* CreaLista(char* ElencoCaratteri);
/* crea in memoria dinamica una lista contenente i caratteri dell'array
 * ElencoCaratteri. L'ultimo elemento della lista è quello che contiene il
 * carattere che precede il '\0' che ElencoCaratteri deve contenere.
 * Restituisce un puntatore al primo elemento della lista. */

void StampaElementi(ElemLista* pTesta);
/* riceve la testa di una lista e ne stampa il campo Dato degli elementi */

void InvertiElementi(ElemLista** ppTesta);
/* riceve un puntatore alla testa di una lista e inverte l'ordine degli
 * elementi della lista; si noti che riceve la testa PER INDIRIZZO in
 * modo da poterne modificare il valore */


int main()
{
  ElemLista* pTesta;
  /* testa della lista */

  pTesta = CreaLista("Lista di prova!");
```

```
  /* NOTA: il compilatore aggiunge automaticamente un carattere '\0' in coda
   * alle stringhe indicate esplicitamente tra doppi apici, come questa;
   * perciò non c'è bisogno di fare altro per ottenere un array con il
   * formato richiesto dalla funzione CreaLista */

  printf("\n\nLista iniziale:\n\n");
  StampaElementi(pTesta);
  printf("\n\n");

  InvertiElementi(&pTesta);

  printf("\n\nLista invertita:\n\n");
  StampaElementi(pTesta);
  printf("\n\n");

  return(0);
}


ElemLista* CreaLista(char* ElencoCaratteri)
{
  int Cont;
  /* usato per scorrere ElencoCaratteri */
  ElemLista* pTestaLista;
  /* testa della lista, da restituire al programma chiamante */
  ElemLista* pElementoCorrente;
  /* punta all'ultimo elemento aggiunto alla lista */

  pTestaLista = NULL;
  Cont = 0;

  while ( '\0' != ElencoCaratteri[Cont] )
  {
    if ( NULL == pTestaLista )   /* la lista è vuota */
          {
            pTestaLista = malloc(sizeof(ElemLista));
    pElementoCorrente = pTestaLista;
    /* ora pElementoCorrente punta all'elemento appena creato */
    pElementoCorrente->Dato = ElencoCaratteri[Cont];
    pElementoCorrente->pProssimo = NULL;
          }
```

```
    else   /* la lista non è vuota */
    {
      pElementoCorrente->pProssimo = malloc(sizeof(ElemLista));
      pElementoCorrente = pElementoCorrente->pProssimo;
      /* ora pElementoCorrente punta all'elemento appena creato */
      pElementoCorrente->Dato = ElencoCaratteri[Cont];
      pElementoCorrente->pProssimo = NULL;
    }

    ++Cont;
  }

  return(pTestaLista);
}


void StampaElementi(ElemLista* pTesta)
{
  if ( NULL != pTesta )
  {
    printf("\t%c", pTesta->Dato);

    StampaElementi(pTesta->pProssimo);
    /* chiamata ricorsiva alla funzione StampaElementi */
  }
}


void InvertiElementi(ElemLista** ppTesta)
{
  /* la funzione legge il contenuto della lista che ha per testa *ppTesta
   * elemento per elemento, e genera una nuova lista. In tale lista vengono
   * inseriti via via, in testa, gli elementi della lista originale letti
   * procedendo verso la coda. Via via che gli elementi della lista originale
   * vengono copiati nella nuova lista, essi vengono deallocati. Infine
   * la testa della lista originale viene collegata alla nuova lista */

  ElemLista* pNuovaTesta;
  /* testa della nuova lista creata */
  ElemLista* pNuovaTestaPrec;
  /* valore precedentemente assunto dalla testa della nuova lista durante
```

```
   * un'operazione di modifica del suo valore */
  ElemLista* pCorrenteVecchia;
  /* puntatore all'elemento corrente della vecchia lista */
  ElemLista* pDaDeallocare;
  /* punta ad un elemento della vecchia lista da deallocare */

  /* creazione degli elementi della nuova lista */
  pNuovaTesta = NULL;
  pCorrenteVecchia = *ppTesta;

  while ( NULL != pCorrenteVecchia )
  /* percorriamo la vecchia lista finche' non abbiamo raggiunto la sua fine;
   * ogni volta che incontriamo un elemento, ne inseriamo uno identico in
   * testa alla nuova lista; infine l'elemento della vecchia lista viene
   * deallocato */
  {
    pNuovaTestaPrec = pNuovaTesta;
    pNuovaTesta = malloc(sizeof(ElemLista));
    pNuovaTesta->Dato = pCorrenteVecchia->Dato;
    pNuovaTesta->pProssimo = pNuovaTestaPrec;

    pDaDeallocare = pCorrenteVecchia;
    pCorrenteVecchia = pCorrenteVecchia->pProssimo;
    free(pDaDeallocare);
  }

  /* NOTA: la deallocazione è indispensabile. Dimenticare di eseguirla e' un
   * errore grave (sebbene il programma funzioni ugualmente). La deallocazione
   * serve ad evitare l'accumulo di "garbage" nella memoria RAM del
   * calcolatore. */

  *ppTesta = pNuovaTesta;
}
```