```
/*
 * Corso di Fondamenti di Informatica
 * Esercizio:
 * prova d'esame del 24 Novembre 2006 - esercizio 1
 */


/*
Si scriva un programma C che esegue la COMPRESSIONE (senza perdita di
informazioni, ovvero "lossless") di un elenco di numeri interi, tutti nulli o
positivi, utilizzando l'algoritmo descritto più avanti.

L'elenco di interi da comprimere va rappresentato tramite una LISTA SEQUENZIALE
(struttura composta da un campo array di 1000 elementi, contenente i dati, ed un
campo intero che indica quanti dati utili sono contenuti nell'array); si chiami
TipoElenco il tipo di dato della lista sequenziale.

Il programma deve definire due variabili di tipo TipoElenco, ElencoOriginale ed
ElencoCompresso. Il contenuto di ElencoOriginale viene determinato da una parte
di programma che NON va scritta (ci si limiti ad un commento), mentre il
programma ha lo scopo di determinare il contenuto di ElencoCompresso.

ElencoCompresso contiene numeri interi di due tipi: (1) quelli positivi o nulli,
che sono dati effettivi prelevati da ElencoOriginale e copiati in
ElencoCompresso; (2) quelli negativi, che chiameremo "riferimenti", ciascuno dei
quali sostituisce un gruppo di 4 interi dell'elenco originale.
[Si noti che ogni introduzione nell'elenco compresso di un riferimento al posto
di un gruppo di 4 interi riduce di 3 interi la lunghezza dell'elenco compresso
rispetto a quella dell'elenco originale, e dunque attua il processo di
compressione. Per ricostruire l'elenco originale a partire da quello compresso,
ovvero per "decomprimere" quest'ultimo, occorrerà sostituire ciascun riferimento
con il gruppo di dati a cui esso fa riferimento. L'esercizio NON richiede la
scrittura del codice C per la decompressione.]

L'algoritmo di creazione dell'elenco compresso è il seguente. L'elenco originale
viene esaminato un elemento alla volta; detto E l'elemento in esame, viene
verificato se il gruppo di 4 interi comprendente E ed i 3 elementi che lo
seguono (detto "gruppo corrente") corrisponde esattamente (anche nell'ordine
degli elementi) ad un gruppo di 4 interi già presente nella parte dell'elenco
compresso che precede il punto di inserimento di E (quest'ultimo gruppo, se
esiste, viene detto "gruppo base"). Se la condizione NON è verificata, E viene
```

```
semplicemente copiato dall'elenco originale all'elenco compresso e si passa ad
esaminare l'elemento successivo dell'elenco originale. Se invece la condizione
è verificata, al posto di E e dei successivi 3 elementi viene inserito
nell'elenco compresso un RIFERIMENTO, ovvero un numero negativo il cui valore
assoluto è pari alla distanza, in elementi dell'elenco compresso, tra il primo
intero del gruppo base ed il punto di inserimento di E; infine si passa ad
esaminare l'elemento posto 4 posizioni dopo E.
Qualora dopo E siano presenti meno di 3 elementi, E va semplicemente copiato
dall'elenco originale a quello compresso, per poi passare ad esaminare
l'elemento successivo dell'elenco originale.

Esempio:
al frammento di ElencoOriginale

  ...4 3 2 1 0 2 4 6 8 1 2 4  2 4 6 8 0 0 0...
              -------        -------

(si noti che il gruppo sottolineato compare due volte) corrisponde, in
ElencoCompresso, il frammento seguente:

  ...4 3 2 1 0 2 4 6 8 1 2 4 -7 0 0 0...

*/


/* NOTA: i commenti di questo programma sono incompleti perché si fa
   riferimento al testo dell'esercizio */
#include <stdio.h>
#define NUM_MAX_INTERI 1000
#define LUNG_GRUPPO 4

typedef enum {false, true} boolean;

typedef struct
{
  int Dati[NUM_MAX_INTERI];
  int NumDatiUtili;
} TipoElenco;

/* NOTA: le definizioni di tipo sono state poste fuori dal blocco main
 * perché sono necessarie anche alle funzioni InputElenco e StampaElenco, non
```

```
 * richieste all'esame ma implementate più in basso*/


void InputElenco(TipoElenco *Elenco);
/* funzione NON RICHIESTA ALL'ESAME per eseguire l'input da tastiera degli
 * elementi di un dato di tipo Tipoelenco */

void StampaElenco(TipoElenco ElDaStamp, char* Stringa);
/* funzione NON RICHIESTA ALL'ESAME per stampare a schermo il contenuto di un
 * dato di tipo Tipoelenco, preceduta dal messaggio Stringa */


int main(void)
{
  TipoElenco ElencoOriginale, ElencoCompresso;
  /* i due elenchi di interi contenenti i dati: quello originale e la sua
   * versione compressa */
  int ContOrig, ContCompr;
  /* contatori usati per scorrere il campo Dati dei due dati di tipo
   * TipoElenco */
  int Incremento;
  /* usato per contare di quanti elementi si è avanzato rispetto ad un elemento
   * di partenza nel campo Dati di ElencoOriginale ed ElencoCompresso */
  boolean Trovato;
  /* vale true se il gruppo di LUNG_GRUPPO interi in esame nell'elenco
   * originale corrisponde al gruppo di LUNG_GRUPPO interi in esame nell'elenco
   * compresso */

  InputElenco(&ElencoOriginale);
  /* creazione dell'elenco originale (NON RICHIESTA ALL'ESAME) */

  StampaElenco(ElencoOriginale, "Elenco originale:\n");
  /* usata per il debugging (NON RICHIESTA ALL'ESAME) */

  ElencoCompresso.NumDatiUtili = 0;
  /* ora ElencoCompresso è vuoto: passiamo al suo riempimento */

  for (ContOrig = 0;
       ContOrig < ElencoOriginale.NumDatiUtili - LUNG_GRUPPO + 1;
       ++ContOrig)
  /* elaboriamo uno dopo l'altro gli elementi dell'elenco originale, verificando
```

```
 * se ciascuno di essi è il primo elemento di un gruppo da comprimere */

/* Nota. I valori di ContOrig considerati da questo ciclo vanno da 0 a
 * ElencoOriginale.NumDatiUtili-LUNG_GRUPPO, ovvero dal primo elemento del
 * campo array di ElencoOriginale all'ultimo elemento che puo' ancora essere
 * il primo di un gruppo composto da LUNG_GRUPPO elementi tutti appartenenti
 * appartenenti all'array ElencoOriginale.Dati.
 * Esempio: se l'array contiene 9 elementi utili, ElencoOriginale.NumDatiUtili
 * vale 10 e il ciclo for considera i valori di ContOrig che vanno da 0 a
 * 10-LUNG_GRUPPO=10-4=6. Come previsto, l'elemento di indice 6 corrisponde
 * proprio al primo elemento dell'ultimo gruppo di LUNG_GRUPPO=4 elementi che
 * e' possibile definire tra gli elementi utili dell'array: quello composto
 * dagli elementi aventi indici 6, 7, 8 e 9. */
{
  Trovato = false;

  for (ContCompr = 0;
       ( (ContCompr < ElencoCompresso.NumDatiUtili - LUNG_GRUPPO + 1) &&
         (false == Trovato) );
             ++ContCompr)
        /* verifichiamo se al gruppo corrente di LUNG_GRUPPO caratteri che ha inizio
         * con l'elemento di indice ContOrig corrisponde nell'elenco compresso un
         * gruppo base che ha inizio con l'elemento di indice ContCompr */
        {
          if (ElencoOriginale.Dati[ContOrig] ==
              ElencoCompresso.Dati[ContCompr])
              /* se l'intero corrente dell'elenco originale compare già
               * nell'elenco compresso */
          {
            /* verifica se quelli che partono dall'indice ContOrig nell'elenco
             * originale e dall'indice ContCompr nell'elenco compresso sono due
             * gruppi di LUNG_GRUPPO elementi identici elemento per elemento */
            Trovato = true;

      for (Incremento = 1;
           (Incremento < LUNG_GRUPPO) && (true == Trovato);
                       ++Incremento)
                {
                  if (ElencoOriginale.Dati[ContOrig+Incremento] !=
                      ElencoCompresso.Dati[ContCompr+Incremento])
                    {
```

```
                    Trovato = false;
                  }
                }

                /* all'uscita dal ciclo for appena concluso, Trovato vale true solo se
                 * e' stato trovato in ElencoCompresso un gruppo base corrispondente al
                 * gruppo corrente attualmente in esame in ElencoOriginale; si noti che
                 * questo fatto provoca l'interruzioone anche del ciclo for piu'
                 * esterno (quello con variabile di controllo ContOrig) */
        }
}

if (true == Trovato)
/* se questa condizione è verificata, l'elemento di indice ContOrig di
 * ElencoOriginale e l'elemento di indice ContCompr di ElencoCompresso
 * sono gli elementi iniziali di due gruppi identici di LUNG_GRUPPO
 * interi: dunque è possibile operare la compressione del gruppo corrente */
     {
       /* inserimento in ElencoCompresso di un riferimento; notare che
        * ElencoCompresso.NumDatiUtili è l'indice al quale verrà inserito
        * il riferimento nell'elenco compresso, mentre ContCompr-1 è
        * l'indice del primo elemento del relativo gruppo base (il -1
        * compensa l'incremento di 1 dato all'uscita dal ciclo for avente
        * ContCompr come contatore) */
       ElencoCompresso.Dati[ElencoCompresso.NumDatiUtili] =
         -1 * (ElencoCompresso.NumDatiUtili - (ContCompr - 1));

       ContOrig = ContOrig + LUNG_GRUPPO - 1;
       /* visto che il riferimento appena inserito rappresenta LUNG_GRUPPO
        * caratteri di ElencoOriginale, anziché uno solo, occorre
        * evitare di esaminare i LUNG_GRUPPO elementi dell'elenco originale
        * che seguono quello correntemente in esame, essendo essi già
        * descritti dal riferimento appena inserito; il -1 e' presente per tenere
        * conto del fatto che al termine dell'iterazione corrente il ciclo for
        * provvedera' comunque ad un incremento di ContOrig pari ad 1  */
     }
else
/* non è possibile effettuare una compressione: in questo caso l'elemento
 * corrente di ElencoOriginale viene semplicemente copiato in
 * ElencoCompresso */
     {
```

```
        ElencoCompresso.Dati[ElencoCompresso.NumDatiUtili] =
                              ElencoOriginale.Dati[ContOrig];
      }

  ++ElencoCompresso.NumDatiUtili;
}

/* se esistono elementi non esaminati di ElencoOriginale (nel qual caso sono
 * meno di LUNG_GRUPPO) li copia in ElencoCompresso */
for (; ContOrig < ElencoOriginale.NumDatiUtili; ++ContOrig)
{
  ElencoCompresso.Dati[ElencoCompresso.NumDatiUtili] =
                             ElencoOriginale.Dati[ContOrig];

  ++ElencoCompresso.NumDatiUtili;
}

StampaElenco(ElencoCompresso, "Elenco compresso:\n");
/* usata per il debugging (NON RICHIESTA ALL'ESAME) */


/* ------------------------------------------------------------*/
/* NOTA: la parte di main che segue non era richiesta all'esame */
/* ------------------------------------------------------------*/

printf("\nDecomprimendo l'elenco compresso si ottiene:\n");

for (ContCompr = 0; ContCompr < ElencoCompresso.NumDatiUtili; ++ContCompr)
{
  if (ElencoCompresso.Dati[ContCompr] >= 0)
      {
        printf("%d\t", ElencoCompresso.Dati[ContCompr]);
      }
  else
      {
        for (Incremento = 0; Incremento < LUNG_GRUPPO; ++Incremento)
        {
          printf("%d\t", ElencoCompresso.Dati
                         [ContCompr+ElencoCompresso.Dati[ContCompr]+Incremento]);
        }
      }
```

```
  }

  printf("\n\n");

  return 0;
}


/*****************************************************************/
/**** NOTA: tutto ciò che segue NON era richiesto all'esame! *****/
/*****************************************************************/

void InputElenco(TipoElenco *Elenco)
{
  boolean Uscita;
  /* vale false fino a che l'input non è stato terminato dall'utente */

  printf ("\n\nInserisci max %d interi >=0. Se <0 l'inserimento termina.\n",
          NUM_MAX_INTERI);

  Elenco->NumDatiUtili = 0;
  Uscita = false;

  do
  {
    scanf("%d", &(Elenco->Dati[Elenco->NumDatiUtili]));

    if (Elenco->Dati[Elenco->NumDatiUtili] >= 0)
          {
            ++Elenco->NumDatiUtili;
          }
    else
          {
            Uscita = true;
          }
  } while ((false == Uscita) && (Elenco->NumDatiUtili < NUM_MAX_INTERI - 1));
}


void StampaElenco(TipoElenco ElDaStamp, char* Stringa)
/* funzione NON RICHIESTA per stampare a schermo il contenuto di un dato di
```

```
 * tipo Tipoelenco */
{
  int ContEl;
  /* usato per contare gli elementi già stampati del campo ati di ElDaStamp */

  printf("\n%s", Stringa);

  for (ContEl = 0; ContEl < ElDaStamp.NumDatiUtili; ++ContEl)
  {
    printf("%d\t", ElDaStamp.Dati[ContEl]);
  }
}


/* ESERCIZIO: si modifichi il programma per far svolgere le operazioni di
 * compressione e decompressione a due funzioni, chiamate rispettivamente
 * Comprimi e Decomprimi */
```