```c
/*
 * Corso di Fondamenti di Informatica
 * Esercizio:
 * uso dei dati di tipo boolean, e note sull'uso dei tipi enumerativi
 */


#include <stdio.h>

int main()
{
  float Variabile;
  /* usata nel seguito per alcuni esempi */

  typedef enum {false, true} boolean;
  /* definizione del tipo boolean */

  boolean VarBooleana, AltraVarBooleana;
  /* usate per prove */

  /* in C la valutazione di una espressione che produce un risultato logico
   * (vero/falso) viene svolta in modo tale da restituire un intero che vale
   * 0 se l'espressione è falsa, e 1 se l'espressione è vera. Infatti... */

  printf("\n'true' calcolato = %d", (1<2) );
  /* stampa '1' */
  printf("\n'false' calcolato = %d\n", (1>2) );
  /* stampa '0' */

  /* Attenzione: più precisamente, in C si fa corrispondere il valore logico
   * falso a 0, e il valore logico vero a "nonzero": qualsiasi numero intero
   * diverso da zero (in genere si usa 1, ma non è detto che sia sempre
   * così) */

  /* In C la definizione di un tipo di dato enumerativo fa corrispondere i
   * diversi possibili valori, nell'ordine che hanno nella dichiarazione di
   * tipo, ai possibili valori di un unsigned int: 0, 1, 2, ...
   * Perciò i valori false e true che le variabili di tipo boolean, per come
   * le abbiamo definite noi, possono assumere, vengono mappati dal C sui
   * valori interi 0 e 1 rispettivamente. Lo si può verificare stampando tali
   * variabili come se fossero interi: */
```

```
printf("\n'true' definito dal programmatore = %d", true);
/* stampa '1' */
printf("\n'false' definito dal programmatore = %d", false);
/* stampa '0' */

VarBooleana = (Variabile < 123.45);
/* assegnamento a VarBooleana del valore di una espressione booleana:
 * se Variabile < 123.45 la valutazione della condizione restituisce un 1,
 * e quindi VarBooleana assume il valore true; se Variabile >= 123.45,
 * invece, la condizione vale 0 e a VarBooleana viene dato il valore false.
 * Questo tipo di assegnamento è una delle operazioni che sono consentite
 * dal linguaggio C, ma vanno evitate perché portano a programmi poco chiari
 * (inoltre cosa succederebbe se il particolare compilatore usato usasse un
 * valore "nonzero" diverso da 1 per indicare il false?).
 * Moltro meglio scrivere... */

 if (Variabile < 123.45)
 {
   VarBooleana = true;
 }
 else
 {
   VarBooleana = false;
 }

if ( true == VarBooleana )
/* a volte si trova nei programmi la scrittura (equivalente alla precedente)
 *        if (VarBooleana)
 * E' un'altro esempio di sintassi ammessa dal C ma poco comprensibile */
  {
    printf("Valore 123.45 non raggiunto.");
  }

AltraVarBooleana = 0;
/* altra istruzione che non genera errori di compilazione ma è
 * concettualmente criticabile: ad una variabile di un tipo enumerativo
 * andrebbero dati, in tutti i casi in cui ciò è possibile, solo i valori
 * specificati nella definizione del tipo. Come nel caso... */

AltraVarBooleana = false;   /* corretto */
```

```
  /* e infine: il seguente blocco di codice */
  if ( (VarBooleana && AltraVarBooleana)  ==
       !(VarBooleana || AltraVarBooleana) )
    {
      printf("\nHo usato gli operatori logici AND, OR e NOT.\n");
    }
  /* è sintatticamente corretto e funzionante, ma sicuramente meno chiaro
   * di quello che segue (assolutamente equivalente): */

   if ( ((true == VarBooleana) && (true == AltraVarBooleana))  ==
        !((true == VarBooleana) || (true == AltraVarBooleana)) )
    {
      printf("\nHo usato gli operatori logici AND, OR e NOT.\n");
    }

  return 0;
}
```