Una `ToDoList` rappresenta una lista di compiti da svolgere. Ogni compito è caratterizzato da una descrizione, da una priorità, e dall'informazione se il compito è stato fatto o no. La descrizione di un compito è una stringa di massimo 40 caratteri. La priorità di un compito è un numero maggiore o uguale di uno. Un numero minore indica una priorità maggiore, quindi la priorità massima è uno. Implementare le seguenti operazioni che possono essere effettuate su una `ToDoList`:

--- Metodi invocati nella PRIMA PARTE di main.cpp: ---

✓ **`ToDoList tdl;`**

Costruttore di default che inizializza una `ToDoList`, inizialmente vuota.

✓ **`tdl.aggiungi(descr, prio);`**

Metodo che aggiunge alla `ToDoList` un nuovo compito con descrizione `descr` e priorità `prio`. Il compito è inizialmente da fare. Se uno degli input non è valido o se `descr` non sta in 40 caratteri, il compito non viene aggiunto e la `ToDoList` rimane invariata.

✓ **`cout << tdl;`**

Operatore di uscita per il tipo `ToDoList`, che stampa a schermo la lista di compiti nel seguente formato:

```
  1 - Regalo per compleanno Giorgia
V 1 - Fare spesa
  2 - Pulire casa
V 2 - Chiamare Marco
  3 - Pagare bolletta gas
```

Le "V" indicano i compiti già fatti. Il numero ne indica la priorità. A destra del numero si trova la descrizione separata da " - ". I compiti devono essere stampati in ordine di priorità decrescente, e a parità di priorità in ordine temporale di aggiunta. Per esempio, nella `ToDoList` sopra, "`Pulire casa`" e "`Chiamare Marco`" hanno entrambi priorità 2, ma "`Pulire casa`" è stato aggiunto per primo e quindi viene stampato per primo.

✓ **`~ToDoList();`**

Distruttore.

--- Metodi invocati nella SECONDA PARTE di main.cpp: ---

✓ **`tdl1 += tld2;`**

Operatore di somma e assegnamento tra due `ToDoList`, che aggiunge alla lista a sinistra tutti i compiti contenuti nella lista a destra. Tali compiti conservano le descrizioni e le priorità che avevano nella lista a destra, e sono tutti da fare.

✓ **`tdl.fai(descr);`**

Funzione che cambia lo stato del compito con descrizione `descr` in "fatto", se tale compito esiste. Se nella lista ci sono più compiti con la stessa descrizione `descr`, viene messo a "fatto" il primo che non è stato ancora fatto, seguendo lo stesso ordine dell'output (vedi operatore `cout<<tdl`).

✓ **`tdl.cancella_fatti();`**

Funzione che rimuove dalla `ToDoList` tutti i compiti fatti.

Mediante il linguaggio C++, realizzare il tipo di dato astratto **`ToDoList`**, definito dalle precedenti specifiche. Non è permesso utilizzare funzionalità della libreria STL come il tipo `string`, il tipo `vector`, il tipo `list`, ecc. **Gestire le eventuali situazioni di errore**.

USCITA CHE DEVE PRODURRE IL PROGRAMMA

```
--- PRIMA PARTE ---
Test costruttore e funzione aggiungi
  1 - Task3
  2 - Task1
  2 - Task2
  2 - Task5
  3 - Task4

Test distruttore
Distruttore chiamato
--- SECONDA PARTE ---
Test operatore +=
  1 - Task3
  1 - Task1
  2 - Task1
  2 - Task2
  2 - Task5
  2 - Task2
  3 - Task4
  3 - Task3
  4 - Task4

Test funzione fai
  1 - Task3
V 1 - Task1
  2 - Task1
V 2 - Task2
  2 - Task5
V 2 - Task2
  3 - Task4
  3 - Task3
  4 - Task4

Test funzione cancella_fatti
  1 - Task3
  2 - Task1
  2 - Task5
  3 - Task4
  3 - Task3
  4 - Task4
```

Note per la consegna:

Affinché l'elaborato venga considerato valido, il programma **deve** produrre almeno la prima parte dell'output atteso. In questo caso, i docenti procederanno alla valutazione dell'elaborato **solo se** lo studente avrà completato l'autocorrezione del proprio elaborato.
In **tutti** gli altri casi (per esempio, il programma non compila, non collega, non esegue o la prima parte dell'output non coincide con quella attesa), l'elaborato è considerato **insufficiente** e, pertanto, **non verrà corretto**.