Da codice etico unisa http://web.unisa.it/uploads/rescue/41/76/codice-etico-e-di-comportamento-unisa.pdf
A**RT. 43 – VIOLAZIONE DEI DOVERI DEL CODICE - STUDENTI**
1. La violazione delle norme del presente Codice da parte degli studenti può dar luogo a sanzioni disciplinari, ai sensi del Regolamento Studenti dell'Ateneo.
2. Quando siano accertate attività tese a modificare indebitamente l'esito delle prove o impedirne una corretta valutazione, il docente o altro preposto al controllo dispone l'annullamento delle prove medesime e la segnalazione al Rettore ai fini dell'attivazione del procedimento disciplinare ai sensi del Regolamento studenti.

Da Regolamento studenti unisa http://web.unisa.it/uploads/rescue/31/19/reg_studenti_2014_web.pdf
**ART. 40 – SANZIONI DISCIPLINARI A CARICO DEGLI STUDENTI**
1. Le sanzioni che si possono comminare sono le seguenti:
a) ammonizione;
b) interdizione temporanea da uno o più attività formative;
c) esclusione da uno o più esami o altra forma di verifica di profitto per un periodo fino a sei mesi;
d) sospensione temporanea dall'Università con conseguente perdita delle sessioni di esame.
2. La relativa competenza è attribuita al Senato accademico, fatto salvo il diritto dello studente destinatario del provvedimento di essere ascoltato.
3. L'applicazione delle sanzioni disciplinari deve rispondere a criteri di ragionevolezza ed equità, avuto riguardo alla natura della violazione, allo svolgimento dei fatti e alla valutazione degli elementi di prova. Le sanzioni sono comminate in ordine di gradualità secondo la gravità dei fatti.
4. La sanzione è comminata con decreto rettorale.
**5. Tutte le sanzioni disciplinari sono registrate nella carriera scolastica dello studente e vengono conseguentemente trascritte nei fogli di congedo.**

## Istruzioni: compilazione

È richiesto scrivere il *makefile* e che il progetto compili con il comando *make*.
Nel caso non si riesca a realizzare un *makefile* corretto è possibile compilare richiamando da linea di comando *compila.bat*. In tal caso ciò **andrà indicato mettendo un commento nel makefile**; si avrà inoltre una penalizzazione sulla valutazione.

## Istruzioni: commenti nel codice

È necessario inserire commenti nel codice prodotto. In particolare, è necessario:

- descrivere la funzione realizzata ed il suo funzionamento con un commento inserito immediatamente prima l'intestazione della funzione; è inoltre necessario scrivere quanto eventualmente richiesto nello specifico dalla traccia.
- commentare i punti più importanti del codice con commenti inline.

Esercizio 1. Si completi il progetto fornito (Esercizio1).
Sviluppare un nuovo operatore di Playlist che dato in input una playlist la ordini dalla canzone più gradita alla meno gradita. Sviluppare un algoritmo di ordinamento a scelta del candidato. Non modificare l'ADT list.

Completare ed estendere quando necessario il codice fornito, in particolare per gestire per ogni canzone anche il gradimento. Si ricorda che l'ADT Playlist è in grado di gestire una lista di canzoni rappresentate tramite l'ADT Song.

Si indichi anche, utilizzando i commenti nel codice, la specifica sintattica e semantica e la complessità asintotica dell'operatore implementato, fornendo una breve giustificazione sulla risposta. **ATTENZIONE:** leggere le note sui commenti e la compilazione nella prima pagina della traccia.

**ATTENZIONE: NON modificare item.h, list.h, list.c. Soluzioni che vadano a modificare tali file non verranno prese in considerazione.**

**ATTENZIONE**: l'operatore deve fare stampe.

**ATTENZIONE: affinché l'esercizio venga valutato è necessario che il progetto modificato compili (con il comando make). La valutazione è fortemente influenzata dal fatto che il progetto modificato esegua senza errori (ad es. segmentation fault) e produca un output.**

Completare `main()` in modo da testare l'operatore sulle seguenti playlist (NON chiedere input all'utente da tastiera, inserire tutto nel codice C). Vedi esempio di output di più in basso per il contenuto della playlist.

1. Titolo playlist: "Playlist vuota"
2. Titolo playlist: "Playlist con una canzone"
3. Titolo playlist: "Playlist con due canzoni"
4. Titolo playlist: "Playlist con quattro canzoni"
5. Titolo playlist: "Playlist con cinque canzoni con gradimento casuale"

In `main()` utilizzare le funzioni di supporto fornite all'inizio di main.c

Esempio di output: **(NOTA: il vostro programma DEVE riprodurre questo output, eccetto l'ultimo caso in cui i gradimenti devono essere differenti ad ogni esecuzione del programma)**

```
-- Stampa playlist originale --
Playlist: Playlist vuota

-- Stampa playlist dopo ordinamento per gradimento --
Playlist: Playlist vuota



-- Stampa playlist originale --
Playlist: Playlist con una canzone
artista A - titolo A (180 sec.) - gradimento: 80

-- Stampa playlist dopo ordinamento per gradimento --
Playlist: Playlist con una canzone
artista A - titolo A (180 sec.) - gradimento: 80
```

```
-- Stampa playlist originale --
Playlist: Playlist con due canzoni
artista B - titolo B (180 sec.) - gradimento: 70
artista C - titolo C (180 sec.) - gradimento: 80

-- Stampa playlist dopo ordinamento per gradimento --
Playlist: Playlist con due canzoni
artista C - titolo C (180 sec.) - gradimento: 80
artista B - titolo B (180 sec.) - gradimento: 70
```

```
-- Stampa playlist originale --
Playlist: Playlist con quattro canzoni
artista E - titolo E (180 sec.) - gradimento: 40
artista F - titolo F (180 sec.) - gradimento: 90
artista G - titolo G (180 sec.) - gradimento: 60
artista H - titolo H (180 sec.) - gradimento: 60

-- Stampa playlist dopo ordinamento per gradimento --
Playlist: Playlist con quattro canzoni
artista F - titolo F (180 sec.) - gradimento: 90
artista G - titolo G (180 sec.) - gradimento: 60
artista H - titolo H (180 sec.) - gradimento: 60
artista E - titolo E (180 sec.) - gradimento: 40
```

```
-- Stampa playlist originale --
Playlist: Playlist con cinque canzoni con gradimento casuale
artista J - titolo J (180 sec.) - gradimento: 76
artista K - titolo K (180 sec.) - gradimento: 14
artista L - titolo L (180 sec.) - gradimento: 36
artista M - titolo M (180 sec.) - gradimento: 1
artista N - titolo N (180 sec.) - gradimento: 100

-- Stampa playlist dopo ordinamento per gradimento --
Playlist: Playlist con cinque canzoni con gradimento casuale
artista N - titolo N (180 sec.) - gradimento: 100
artista J - titolo J (180 sec.) - gradimento: 76
artista L - titolo L (180 sec.) - gradimento: 36
artista K - titolo K (180 sec.) - gradimento: 14
artista M - titolo M (180 sec.) - gradimento: 1
```

Esercizio 2. Si implementi, completando il progetto fornito (Esercizio2), due funzioni:

- **heightAndNumNodes** che prende in input un albero binario e che calcola la sua altezza ed il suo numero di nodi (la funzione restituisce un array di due elementi, il primo contenente l'altezza ed il secondo il numero di nodi). Usare la tecnica della **ricorsione** e scorrere l'albero **una sola volta** per calcolare entrambi i valori (sia altezza che numero dei nodi - **non** calcolare separatamente i due valori).
- **preorder** che prende in input un albero binario e stampa in maniera **iterativa** una visita PreOrder dell'albero.

Si indichi anche, utilizzando i commenti nel codice, la complessità asintotica delle funzioni implementate, fornendo una breve giustificazione sulla risposta. **ATTENZIONE:** leggere le note sui commenti e la compilazione nella prima pagina della traccia.

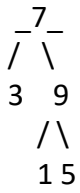
**Modificare esclusivamente il file main.c** (completando le funzioni già presenti ed eventualmente aggiungendone altre se necessario) **ed il makefile, senza aggiungere o modificare altri file. NON toccare BTree. Soluzioni non realizzate in main.c non verranno prese in considerazione.**

**ATTENZIONE**: la funzione **heightAndNumNodes** non deve fare stampe.

**ATTENZIONE: affinché l'esercizio venga valutato è necessario che il progetto modificato compili (con il comando make). La valutazione è fortemente influenzata dal fatto che il progetto modificato esegua senza errori (ad es. segmentation fault) e produca un output.**

Testare nel main la procedura sui seguenti alberi (NON chiedere input all'utente da tastiera, inserire tutto nel codice C):

- Un albero vuoto;
- Un albero consistente della sola radice contenente il numero 2;
- Un albero instanziato come di seguito




- Un albero casuale con 8 nodi

Utilizzare la funzione **newRandomTree(int nNodes)** per creare l'albero casuale (fornita dall'ADT) e la funzione **printHeightNumNodesAndPreorder(BTree)** per stampare altezza, numero nodi e visita preorder dell'albero.

Esempio di output: **(NOTA: il vostro programma deve riprodurre questo output)**

```
Albero:
Altezza albero: 0
Numero nodi albero: 0
Visita preorder iterativa:

Albero:
2
Altezza albero: 0
```

```
Numero nodi albero: 1
Visita preorder iterativa: 2

Albero:
   _7_
  /   \
  3   9
     / \
     1 5
Altezza albero: 2
Numero nodi albero: 5
Visita preorder iterativa: ... lasciata allo studente

Albero:
      ___9___
    /       \
   _7_      _8_
  /   \       \
  3   0        2
   \          /
   9          0
Altezza albero: 3
Numero nodi albero: 8
Visita preorder iterativa: ... lasciata allo studente
```

Esercizio 3. Si implementi, completando il progetto fornito (Esercizio3), una **procedura** che modifica il contenuto di uno stack invertendo l'ordine degli elementi in esso contenuti, e lasciando **solo gli elementi maggiori ad un dato elemento fornito come parametro** (vedere esempio di output). **Modificare esclusivamenti i file main.c e makefile** aggiungendo le funzioni necessarie e completando `int main()`. Utilizzare i soli operatori di base dello stack (non modificare stack.c e stack.h). Non restituire un nuovo stack.

Si indichi anche, utilizzando i commenti nel codice, la complessità asintotica della procedura implementata, fornendo una breve giustificazione sulla risposta. **ATTENZIONE:** leggere le note sui commenti e la compilazione nella prima pagina della traccia.

Testare la procedura sui seguenti **parametro/coda leggendo l'input da file input.txt (fornito) dove ogni riga corrisponde al parametro seguito dal contenuto di una lista** (NON chiedere input all'utente da tastiera):

- Parametro: "ciao"; Stack: stack_vuoto
- Parametro: "ciao"; Stack: "ciao"
- Parametro: "ciao"; Stack (ordine di inserimento): "ciao", "bella", "gente", "come", "va"
- Parametro: "dado"; Stack (ordine di inserimento): "ciao", "bella", "gente", "come", "va"
- Parametro: item_casuale; Stack: 7 item causali
  (per generare gli item casuali usare `randomItem()`). **NB: nel file input.txt l'elemento casuale corrisponde a ! . Quindi quando si trova tale elemento va inserito nella lista un elemento casuale con randomItem().**

**ATTENZIONE**: la procedura non deve fare stampe.

**ATTENZIONE: affinché l'esercizio venga valutato è necessario che il progetto modificato compili (con il comando make). La valutazione è fortemente influenzata dal fatto che il progetto modificato esegua senza errori (ad es. segmentation fault) e produca un output.**

Esempio di output: **(NOTA: il vostro programma DEVE riprodurre questo output, eccetto per il parametro casuale e lo stack casuale che dovranno essere differenti ad ogni esecuzione del programma)**

```
Elemento parametro: ciao
Stack originale:
Stack invertito:

Elemento parametro: ciao
Stack originale: ciao
Stack invertito:

Elemento parametro: ciao
Stack originale: va     come  gente bella ciao
Stack invertito: gente  come  va

Elemento parametro: dado
Stack originale: va     come  gente bella ciao
Stack invertito: gente  va

Elemento parametro: dep
Stack originale: fbj    zdm   d     rri   cjk   kwi   ccp
Stack invertito: kwi    rri   zdm   fbj
```

*Di seguito è presente una versione alternativa dell'esercizio 3.*

Esercizio 3b. Si implementi, completando il progetto fornito (Esercizio3b), una **procedura** che modifica il contenuto di uno stack invertendo l'ordine degli elementi in esso contenuti, e lasciando **solo gli elementi maggiori ad un dato elemento fornito come parametro** (vedere esempio di output). **Modificare esclusivamenti i file main.c e makefile** aggiungendo le funzioni necessarie e completando `int main()`. Utilizzare i soli operatori di base dello stack (non modificare stack.c e stack.h). Non restituire un nuovo stack.

Si indichi anche, utilizzando i commenti nel codice, la complessità asintotica della procedura implementata, fornendo una breve giustificazione sulla risposta. **ATTENZIONE:** leggere le note sui commenti e la compilazione nella prima pagina della traccia.

Testare la procedura sui seguenti **parametro/stack** (NON chiedere input all'utente da tastiera, inserire tutto nel codice C):

- Parametro: "ciao"; Stack: stack_vuoto
- Parametro: "ciao"; Stack: "ciao"
- Parametro: "ciao"; Stack (ordine di inserimento): "ciao", "bella", "gente", "come", "va"
- Parametro: "dado"; Stack (ordine di inserimento): "ciao", "bella", "gente", "come", "va"
- Parametro: item_casuale; Stack: 7 item causali
  (per generare gli item casuali usare `randomItem()`)

**ATTENZIONE**: la procedura non deve fare stampe.

**ATTENZIONE: affinché l'esercizio venga valutato è necessario che il progetto modificato compili (con il comando make). La valutazione è fortemente influenzata dal fatto che il progetto modificato esegua senza errori (ad es. segmentation fault) e produca un output.**

Esempio di output: **(NOTA: il vostro programma DEVE riprodurre questo output, eccetto per il parametro casuale e lo stack casuale che dovranno essere differenti ad ogni esecuzione del programma)**

```
Elemento parametro: ciao
Stack originale:
Stack invertito:

Elemento parametro: ciao
Stack originale: ciao
Stack invertito:

Elemento parametro: ciao
Stack originale: va     come  gente bella ciao
Stack invertito: gente  come  va

Elemento parametro: dado
Stack originale: va     come  gente bella ciao
Stack invertito: gente  va

Elemento parametro: dep
Stack originale: fbj    zdm   d     rri   cjk   kwi   ccp
Stack invertito: kwi    rri   zdm   fbj
```