Esame 20220111

Esercizio 1

(1) **Esercizio 1 v1**

ESSAY | marked out of 10 | penalty 0 | File picker

Scrivere nel file `esercizio1.cc` programma che, presi come argomento del main:

- Un file contenente una lista non ordinata di voti compresi tra 18 e 30.
- Un intero `DIM` che rappresenta il numero totale di elementi presenti nel file di input.
- Un interno `N`, minore o uguale a `DIM`.

crei un nuovo file di testo (chiamato `output.txt`) contenente una lista ordinata con gli `N` voti piú alti presenti nel file passato da riga di comando.

Supponiamo che il primo file `input.txt` contenga

```
23
30
18
27
20
21
18
25
```

allora il comando

```
./esercizio1.out input.txt 8 3
```

e dovrá produrre un file chiamato `output.txt` che conterrá i seguenti valori:

```
30
27
25
```

Note:

- Utilizzare la funzione `sort_array()` fornita per effettuare l'ordinamento.
- Non é concesso fare assunzioni sul numero **massimo** di elementi presenti nel file pena l'annullamento dell'esercizio.
- É consentito l'utilizzo della funzione `int atoi (const char * str)`.
- Non é consentito l'utilizzo di altre funzioni di libreria "particolari" diverse da quelle specificate sopra o da quelle standard necessarie a risolvere l'esercizio.
- Non é consentito inizializzare array dinamici con la sintassi `int array[n]`, dove `n` é una variabile non definita a compile time.
- Le uniche assunzioni che si possono fare sull'input e su dimensioni di eventuali strutture/array utilizzate nel file di partenza fornito sono **solo quelle espressamente specificate in questo testo** (e NON quelle riportate nel file fornito, che sono SOLO indicative per consentire di svolgere l'esame).

- All'interno di questo programma **non è ammesso** l'utilizzo di variabili globali o di tipo `static`.
- Si ricorda che, l'esempio di esecuzione è puramente indicativo, e la soluzione proposta NON deve funzionare solo per l'input fornito, ma deve essere robusta a variazioni compatibili con la specifica riportata in questo testo.

Information for graders:

Esame 20220111

Esercizio 2

(1) **Esercizio 2 v1**

ESSAY | marked out of 10 | penalty 0 | File picker

Scrivere la dichiarazione e la definizione di una procedura **ricorsiva** `compute_lists`, che prende come argomento una stringa (letta da tastiera) di lunghezza non nota e che non può essere modificata (i.e. `const`), ma ben formata (i.e. con carattere di terminatore `'\0'`), e due liste concatenate `s1` e `s2` di tipo `node *` passate entrambe per riferimento. La procedura ricorsiva deve fare le seguenti operazioni:

- Ogni carattere compreso tra `'A'` e `'M'` (inclusi) viene trasformato in modo che `'A'` diventi `'m'`, `'B'` diventi `'l'`, ..., `'L'` diventi `'b'`, e `'M'` diventi `'a'`, il carattere risultante viene inserito nella lista `s1` in modo che l'ordine in cui si incontrano rispetti l'ordine della stringa di partenza;
- Ogni carattere compreso tra `'N'` e `'Z'` (inclusi) viene trasformato in modo che `'N'` diventi `'z'`, `'Ò` diventi `'y'`, ..., `'Y'` diventi `'ò`, e `'Z'` diventi `'n'`, il carattere risultante viene inserito nella lista `s2` in modo che l'ordine in cui si incontrano rispetti l'ordine della stringa di partenza;
- Ogni carattere che non soddisfa le regole di cui sopra é ignorato.

La procedura `compute_list` può solo chiamare se stessa, e solo con lo stesso numero di argomenti (NON è consentito fare overloading e/o utilizzare argomenti addizionali inizializzati nella dichiarazione/definizione). Sono solo consentite (se ritenute necessarie) chiamate a funzioni semplici per facilitare/rendere modulare la conversione di un carattere in altro carattere per realizzare le condizioni di cui sopra. La conversione dei caratteri deve essere realizzata SOLO mediante operazioni aritmetiche sui caratteri. NON sono ammesse soluzioni che usano case/switch/if per effettuare le conversioni di cui sopra.

La funzione è inserita in un semplice programma che legge una stringa da tastiera, costruisce le due liste concatenate, le stampa, e dealloca le liste create. Il `main`, le funzioni `stampa_lista`, e `dealloca` NON devono essere modificate. Alcuni esempi di esecuzione sono i seguenti:

```
computer > ./a.out "-=abcdefghijklmnopqrstuvwxyz0123456789."
La stringa da analizzare e': -=abcdefghijklmnopqrstuvwxyz0123456789.
Lista s1:
Lista s2:
computer > ./a.out "-=ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789."
La stringa da analizzare e': -=ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789.
Lista s1: m l k j i h g f e d c b a
Lista s2: z y x w v u t s r q p o n
```

Note:

- Scaricare il file `esercizio2.cc`, modificarlo per inserire la dichiarazione e la definizione della funzione `compute_list`, e caricare il file sorgente risultato delle vostre modifiche a soluzione di questo esercizio nello spazio apposito.

- La funzione `compute_list` deve essere ricorsiva ed al suo interno **NON ci possono essere cicli o chiamate a funzioni contenenti cicli**. Si può però fare uso di funzioni ausiliarie da chiamare all'interno di questa funzione che NON contengano cicli secondo le restrizioni specificate sopra.
- Le uniche assunzioni che si possono fare sull'input e su dimensioni di eventuali strutture/array utilizzate nel file di partenza fornito sono **solo quelle espressamente specificate in questo testo** (e NON quelle riportate nel file fornito, che sono SOLO indicative per consentire di svolgere l'esame).
- All'interno di questo programma **non è ammesso** l'utilizzo di variabili globali o di tipo `static` e di funzioni di libreria al di fuori di quelle definite in `cstddef` o `cstdlib`.
- Si ricorda che, l'esempio di esecuzione è puramente indicativo, e la soluzione proposta NON deve funzionare solo per l'input fornito, ma deve essere robusta a variazioni compatibili con la specifica riportata in questo testo.

Information for graders:

Esame 20220223

Esercizio 3

(1) **Esercizio 3 v1**

ESSAY | marked out of 10 | penalty 0 | File picker

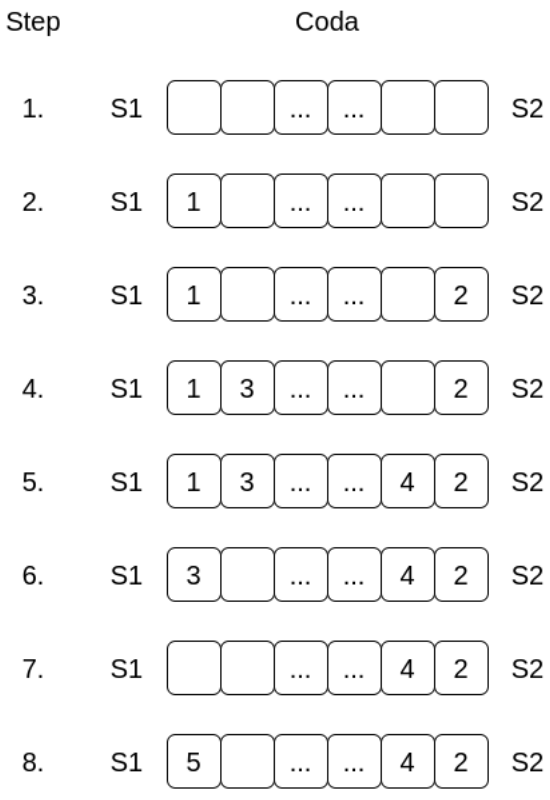
In un ufficio postale, ci sono due sportelli attivi per servire i clienti. All'arrivo, ogni cliente viene identificato con un numero e si mette in fila allo sportello con meno clienti. I clienti non cambiano mai fila. Il massimo numero di clienti all'interno dell'ufficio postale è fissato a 11.

Scrivere nel file `coda.cc` l'implementazione di una variante di una coda per lo scenario sopra descritto. La coda deve tenere traccia dei clienti in fila ai due sportelli—devono esistere due "sotto-code" chiamate S1 e S2, una per ogni sportello—e aggiungere nuovi clienti alla sotto-coda con meno clienti. **Importante**: la coda deve essere implementata usando uno ed un solo array (oppure una e una sola lista concatenata). La coda deve implementare le seguenti funzioni (se presente, il valore di ritorno è "true" se l'operazione è andata a buon fine, "false" altrimenti):

- $void\ init()$: inizializza la coda e altri valori rilevanti, se necessario;
- $bool\ enqueue(int)$: inserimento di un elemento nella sotto-coda con meno elementi, se il numero massimo di elementi non è stato raggiunto;
- $bool\ firstS1(int\&)$: assegna al parametro il valore del primo elemento di S1, se presente;
- $bool\ firstS2(int\&)$: assegna al parametro il valore del primo elemento di S2, se presente;
- $bool\ dequeueS1()$: rimuove il primo elemento di S1, se presente;
- $bool\ dequeueS2()$: rimuove il primo elemento di S2, se presente;
- $void\ deinit()$: de-inizializza la coda e dealloca eventuale memoria dinamica, se necessario;
- $void\ print()$: stampa a video tutti gli elementi di S1 e S2.

Questo è un esempio di esecuzione con a lato una rappresentazione grafica della coda:

```
computer > ./a.out
[Step 1] Digita il comando
Il tuo comando: aggiungi
[Step 2] Abbiamo inserito 1
Il tuo comando: aggiungi
[Step 3] Abbiamo inserito 2
Il tuo comando: aggiungi
[Step 4] Abbiamo inserito 3
Il tuo comando: aggiungi
[Step 5] Abbiamo inserito 4
Il tuo comando: stampa
S1: 1 3
S2: 2 4
Il tuo comando: rimuoviS1
[Step 6] abbiamo rimosso 1
Il tuo comando: rimuoviS1
[Step 7] abbiamo rimosso 3
Il tuo comando: aggiungi
[Step 8] Abbiamo inserito 5
Il tuo comando: p
S1: 5
S2: 2 4
```



Note:

- Creare un file dal nome `coda.cc` e scrivere dentro al file l'implementazione della coda, come da consegna. E' possibile scaricare i file `esercizio3.cc` (che contiene un main di prova) e `coda.h` (che contiene la definizione delle funzioni da implementare) per testare il codice scritto. Infine, caricare solo il file `coda.cc` nello spazio apposito;
- Non modificare i file `esercizio3.cc` e `coda.h`. In altre parole, l'implementazione della coda **deve** essere compatibile con il codice scritto in `esercizio3.cc` e `coda.h`, poiché questi due file verranno usati per valutare la correttezza dell'esercizio;
- All'interno del file `coda.cc` non è ammesso l'utilizzo di funzioni di libreria al di fuori di quelle definite in `iostream`;
- Ricordarsi di distinguere gli esempi nella descrizione dell'esercizio (che servono solo ad aiutare a comprendere il problema) dalle istruzioni di implementazione.

Suggerimenti:

- E' possibile partire dall'implementazione di una coda "tradizionale" e modificare il codice per soddisfare la consegna dell'esercizio.

Information for graders: