Esame 20220223

Esercizio 3

(1) **Esercizio 3 v1**

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

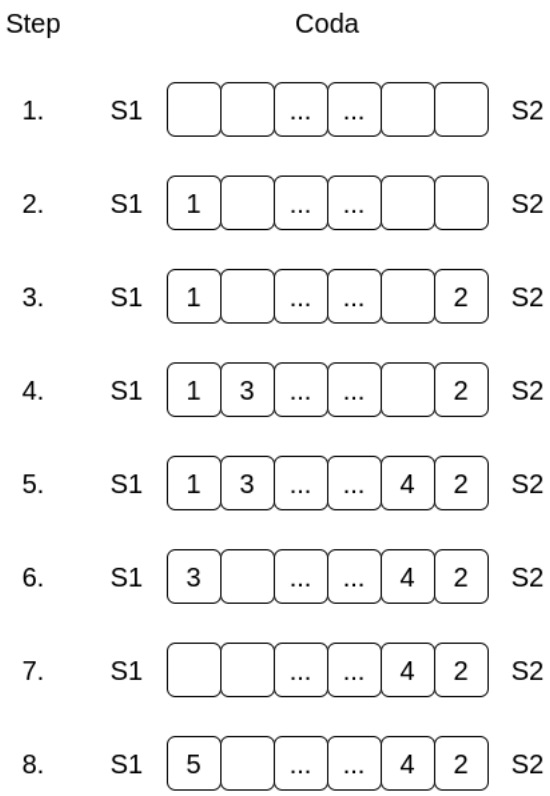
In un ufficio postale, ci sono due sportelli attivi per servire i clienti. All'arrivo, ogni cliente viene identificato con un numero e si mette in fila allo sportello con meno clienti. I clienti non cambiano mai fila. Il massimo numero di clienti all'interno dell'ufficio postale è fissato a 11.

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 variabili globali e 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:

(2) **Esercizio 3 v2**

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

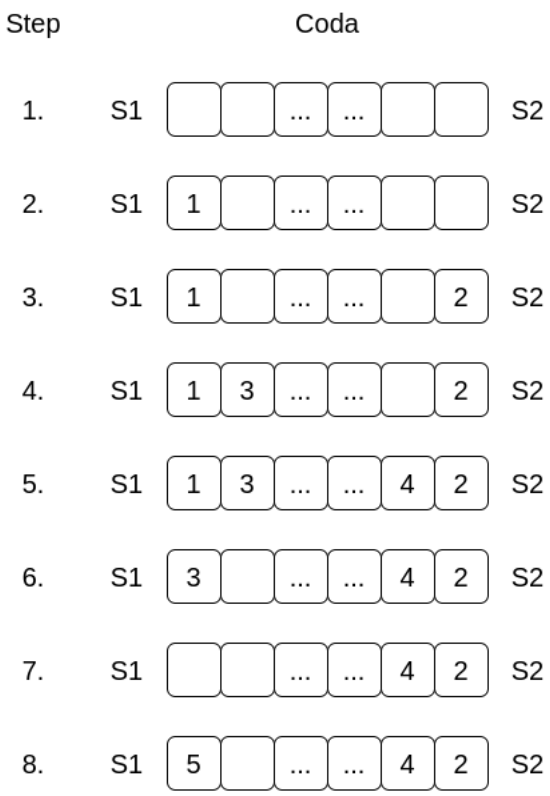
In un ufficio postale, ci sono due sportelli attivi per servire i clienti. All'arrivo, ogni cliente viene identificato con un numero e si mette in fila allo sportello con meno clienti. I clienti non cambiano mai fila. Il massimo numero di clienti all'interno dell'ufficio postale è fissato a 13.

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 variabili globali e 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:

(3) **Esercizio 3 v3**

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

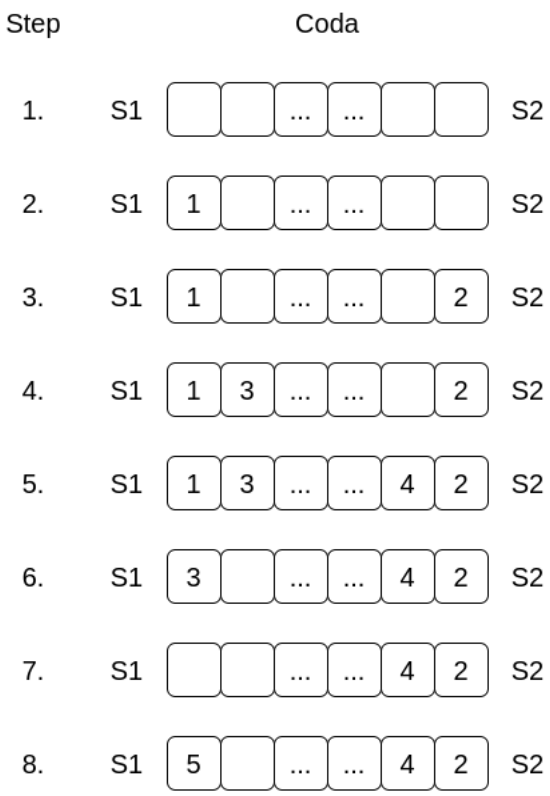
In un ufficio postale, ci sono due sportelli attivi per servire i clienti. All'arrivo, ogni cliente viene identificato con un numero e si mette in fila allo sportello con meno clienti. I clienti non cambiano mai fila. Il massimo numero di clienti all'interno dell'ufficio postale è fissato a 15.

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 variabili globali e 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:

(4) **Esercizio 3 v4**

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

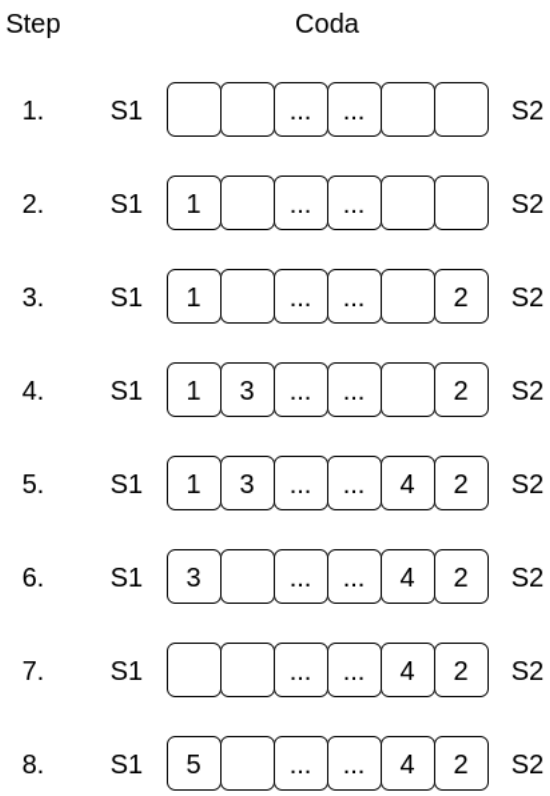
In un ufficio postale, ci sono due sportelli attivi per servire i clienti. All'arrivo, ogni cliente viene identificato con un numero e si mette in fila allo sportello con meno clienti. I clienti non cambiano mai fila. Il massimo numero di clienti all'interno dell'ufficio postale è fissato a 17.

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 variabili globali e 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:

Total of marks: 40