Esame 20230220

Esercizio 3

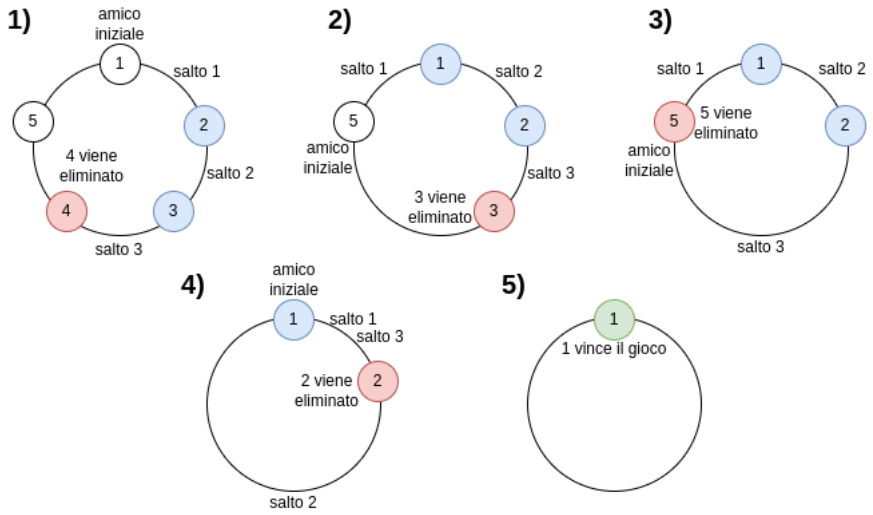
(1) **Esercizio 3 v1**

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

Ci sono `n` amici numerati da `1` a `n` in senso orario che giocano seduti in cerchio. Muoversi (o saltare) dall'amico `i`-esimo in senso orario porta all'amico `(i+1)`-esimo per `1 <= i < n`, mentre saltare dall'amico `n`-esimo in senso orario porta all'amico `1`. Le regole del gioco sono le seguenti:

- partendo dall'amico `1`, ci si muove di un certo numero di salti `k` in senso orario. Alla prima iterazione del gioco, questo significa arrivare fino all'amico `(k % n)+1`-esimo. Notare come, se il numero di salti è maggiore del numero di amici, si gira più volte il cerchio;
- l'ultimo amico a cui si è saltato—alla prima iterazione è l'amico `(k % n)+1`-esimo—viene eliminato e rimosso dal cerchio;
- Se ci sono ancora almeno due amici, il gioco riparte dall'amico immediatamente successivo in senso orario a quello appena eliminato; gli amici rimasti continuano il gioco mantenendo la loro numerazione originale. Altrimenti, l'ultimo amico rimasto vince il gioco.

Questo è un esempio grafico dello svolgimento del gioco con `n = 5` e `k = 3`:



Scrivere nel file `esercizio3.cc` la corretta implementazione della funzione `trovaIlVincitore` che prende come parametri formali un intero `numeroDiAmici` e un intero `numeroDiSalti`. Usando una coda come supporto, la funzione `trovaIlVincitore` deve ritornare il numero dell'amico vincitore del gioco sopra descritto.

Questi sono quattro diversi esempi di esecuzione (i due interi forniti come argomenti da linea di comando sono il numero di amici `n` e il numero di salti `k`, rispettivamente):

```
computer > ./a.out 5 3
Il vincitore e' l'amico numero 1

computer > ./a.out 5 4
Il vincitore e' l'amico numero 2
```

```
computer > ./a.out 4 2
Il vincitore e' l'amico numero 1

computer > ./a.out 4 5
Il vincitore e' l'amico numero 3
```

Note:

- Scaricare il file `esercizio3.cc`, modificarlo per inserire la corretta implementazione della funzione `trovaIlVincitore` e infine caricare il file risultato delle vostre modifiche a soluzione di questo esercizio nello spazio apposito;
- Il numero di amici `n` e il numero di salti `k` sono entrambi numeri interi positivi maggiori di 0;
- Scaricare anche i file `coda.cc` e `coda.h` i quali implementano le funzionalita' di una coda. È obbligatorio usare questi file nella risoluzione dell'esercizio;
- In questo programma, non si possono usare array;
- Ricordarsi di deallocare la memoria allocata dinamicamente e invocare la funzione `deinit()` della coda;
- Ricordarsi di distinguere gli esempi nella descrizione dell'esercizio (che servono solo ad aiutare a comprendere il problema) dalle istruzioni di implementazione;
- È consentito definire ed implementare funzioni ausiliarie che possano aiutarvi nella soluzione del problema;
- 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 `iostream`.

coda.h
coda.cc
esercizio3.cc

Information for graders:

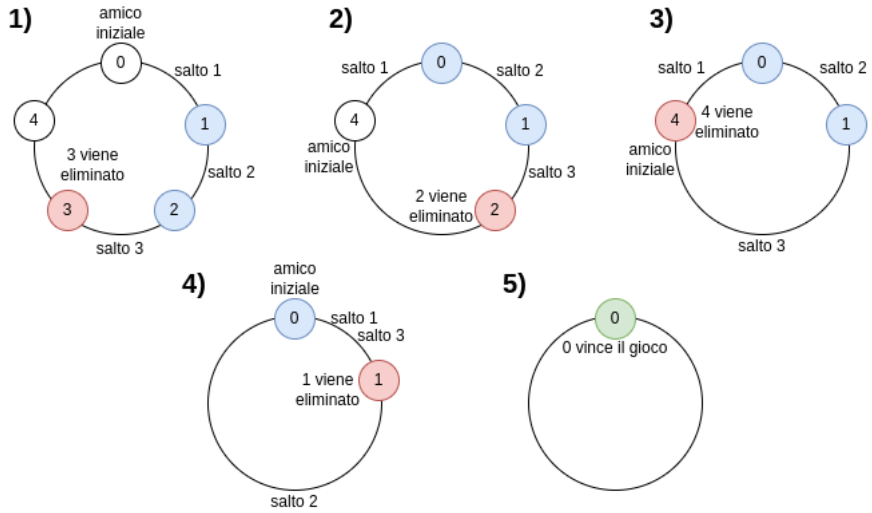
(2) **Esercizio 3 v2**

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

Ci sono `n` amici numerati da `0` a `n-1` in senso orario che giocano seduti in cerchio. Muoversi (o saltare) dall'amico `i`-esimo in senso orario porta all'amico `(i+1)`-esimo per `0 <= i < n-1`, mentre saltare dall'amico `(n-1)`-esimo in senso orario porta all'amico `0`. Le regole del gioco sono le seguenti:

- partendo dall'amico `0`, ci si muove di un certo numero di salti `k` in senso orario. Alla prima iterazione del gioco, questo significa arrivare fino all'amico `(k % n)`-esimo. Notare come, se il numero di salti è maggiore del numero di amici, si gira più volte il cerchio;
- l'ultimo amico a cui si è saltato—alla prima iterazione è l'amico `(k % n)`-esimo—viene eliminato e rimosso dal cerchio;
- Se ci sono ancora almeno due amici, il gioco riparte dall'amico immediatamente successivo in senso orario a quello appena eliminato; gli amici rimasti continuano il gioco mantenendo la loro numerazione originale. Altrimenti, l'ultimo amico rimasto vince il gioco.

Questo è un esempio grafico dello svolgimento del gioco con `n = 5` e `k = 3`:



Scrivere nel file `esercizio3.cc` la corretta implementazione della funzione `trovaIlVincitore` che prende come parametri formali un intero `numeroDiAmici` e un intero `numeroDiSalti`. Usando una coda come supporto, la funzione `trovaIlVincitore` deve ritornare il numero dell'amico vincitore del gioco sopra descritto.

Questi sono quattro diversi esempi di esecuzione (i due interi forniti come argomenti da linea di comando sono il numero di amici `n` e il numero di salti `k`, rispettivamente):

```
computer > ./a.out 5 3
Il vincitore e' l'amico numero 0

computer > ./a.out 5 4
Il vincitore e' l'amico numero 1

computer > ./a.out 4 2
Il vincitore e' l'amico numero 0

computer > ./a.out 4 5
Il vincitore e' l'amico numero 2
```

Note:

- Scaricare il file `esercizio3.cc`, modificarlo per inserire la corretta implementazione della funzione `trovaIlVincitore` e infine caricare il file risultato delle vostre modifiche a soluzione di questo esercizio nello spazio apposito;
- Il numero di amici `n` e il numero di salti `k` sono entrambi numeri interi positivi maggiori di 0;
- Scaricare anche i file `coda.cc` e `coda.h` i quali implementano le funzionalita' di una coda. È obbligatorio usare questi file nella risoluzione dell'esercizio;
- In questo programma, non si possono usare array;
- Ricordarsi di deallocare la memoria allocata dinamicamente e invocare la funzione `deinit()` della coda;
- Ricordarsi di distinguere gli esempi nella descrizione dell'esercizio (che servono solo ad aiutare a comprendere il problema) dalle istruzioni di implementazione;
- È consentito definire ed implementare funzioni ausiliarie che possano aiutarvi nella soluzione del problema;
- 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 `iostream`.

coda.h
coda.cc
esercizio3.cc

Information for graders:

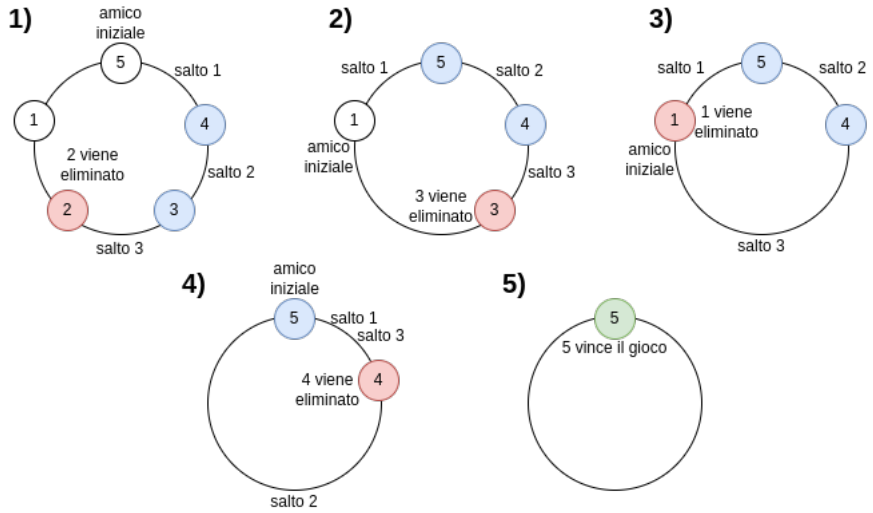
(3) **Esercizio 3 v3**

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

Ci sono `n` amici numerati da `n` a `1` in senso orario che giocano seduti in cerchio. Muoversi (o saltare) dall'amico `i`-esimo in senso orario porta all'amico `(i-1)`-esimo per `1 < i <= n`, mentre saltare dall'amico `1` in senso orario porta all'amico `n`-esimo. Le regole del gioco sono le seguenti:

- partendo dall'amico `n`, ci si muove di un certo numero di salti `k` in senso orario. Alla prima iterazione del gioco, questo significa arrivare fino all'amico `(n- (k % n))`-esimo. Notare come, se il numero di salti è maggiore del numero di amici, si gira più volte il cerchio;
- l'ultimo amico a cui si è saltato—alla prima iterazione è l'amico `(n- (k % n))`-esimo—viene eliminato e rimosso dal cerchio;
- Se ci sono ancora almeno due amici, il gioco riparte dall'amico immediatamente successivo in senso orario a quello appena eliminato; gli amici rimasti continuano il gioco mantenendo la loro numerazione originale. Altrimenti, l'ultimo amico rimasto vince il gioco.

Questo è un esempio grafico dello svolgimento del gioco con `n = 5` e `k = 3`:



Scrivere nel file `esercizio3.cc` la corretta implementazione della funzione `trovaIlVincitore` che prende come parametri formali un intero `numeroDiAmici` e un intero `numeroDiSalti`. Usando una coda come supporto, la funzione `trovaIlVincitore` deve ritornare il numero dell'amico vincitore del gioco sopra descritto.

Questi sono quattro diversi esempi di esecuzione (i due interi forniti come argomenti da linea di comando sono il numero di amici `n` e il numero di salti `k`, rispettivamente):

```
computer > ./a.out 5 3
Il vincitore e' l'amico numero 5

computer > ./a.out 5 4
Il vincitore e' l'amico numero 4

computer > ./a.out 4 2
Il vincitore e' l'amico numero 4

computer > ./a.out 4 5
Il vincitore e' l'amico numero 2
```

Note:

- Scaricare il file `esercizio3.cc`, modificarlo per inserire la corretta implementazione della funzione `trovaIlVincitore` e infine caricare il file risultato delle vostre modifiche a soluzione di questo esercizio nello spazio apposito;
- Il numero di amici `n` e il numero di salti `k` sono entrambi numeri interi positivi maggiori di 0;
- Scaricare anche i file `coda.cc` e `coda.h` i quali implementano le funzionalita' di una coda. È obbligatorio usare questi file nella risoluzione dell'esercizio;
- In questo programma, non si possono usare array;
- Ricordarsi di deallocare la memoria allocata dinamicamente e invocare la funzione `deinit()` della coda;
- Ricordarsi di distinguere gli esempi nella descrizione dell'esercizio (che servono solo ad aiutare a comprendere il problema) dalle istruzioni di implementazione;
- È consentito definire ed implementare funzioni ausiliarie che possano aiutarvi nella soluzione del problema;
- 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 `iostream`.

coda.h
coda.cc
esercizio3.cc

Information for graders:

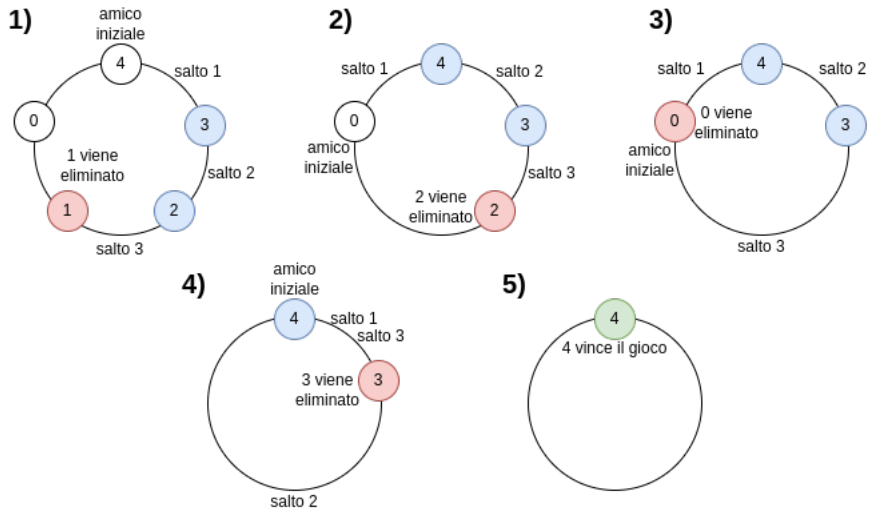
(4) **Esercizio 3 v4**

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

Ci sono `n` amici numerati da `n-1` a `0` in senso orario che giocano seduti in cerchio. Muoversi (o saltare) dall'amico `i`-esimo in senso orario porta all'amico `(i-1)`-esimo per `0 < i <= n-1`, mentre saltare dall'amico `0` in senso orario porta all'amico `(n-1)`-esimo. Le regole del gioco sono le seguenti:

- partendo dall'amico `n-1`, ci si muove di un certo numero di salti `k` in senso orario. Alla prima iterazione del gioco, questo significa arrivare fino all'amico `(n - 1 - (k % n))`-esimo. Notare come, se il numero di salti è maggiore del numero di amici, si gira più volte il cerchio;
- l'ultimo amico a cui si è saltato—alla prima iterazione è l'amico `(n - 1 - (k % n))`-esimo—viene eliminato e rimosso dal cerchio;
- Se ci sono ancora almeno due amici, il gioco riparte dall'amico immediatamente successivo in senso orario a quello appena eliminato; gli amici rimasti continuano il gioco mantenendo la loro numerazione originale. Altrimenti, l'ultimo amico rimasto vince il gioco.

Questo è un esempio grafico dello svolgimento del gioco con `n = 5` e `k = 3`:



Scrivere nel file `esercizio3.cc` la corretta implementazione della funzione `trovaIlVincitore` che prende come parametri formali un intero `numeroDiAmici` e un intero `numeroDiSalti`. Usando una coda come supporto, la funzione `trovaIlVincitore` deve ritornare il numero dell'amico vincitore del gioco sopra descritto.

Questi sono quattro diversi esempi di esecuzione (i due interi forniti come argomenti da linea di comando sono il numero di amici `n` e il numero di salti `k`, rispettivamente):

```
computer > ./a.out 5 3
Il vincitore e' l'amico numero 4

computer > ./a.out 5 4
Il vincitore e' l'amico numero 3

computer > ./a.out 4 2
Il vincitore e' l'amico numero 3
```

```
computer > ./a.out 4 5
Il vincitore e' l'amico numero 1
```

Note:

- Scaricare il file `esercizio3.cc`, modificarlo per inserire la corretta implementazione della funzione `trovaIlVincitore` e infine caricare il file risultato delle vostre modifiche a soluzione di questo esercizio nello spazio apposito;
- Il numero di amici `n` e il numero di salti `k` sono entrambi numeri interi positivi maggiori di 0;
- Scaricare anche i file `coda.cc` e `coda.h` i quali implementano le funzionalita' di una coda. È obbligatorio usare questi file nella risoluzione dell'esercizio;
- In questo programma, non si possono usare array;
- Ricordarsi di deallocare la memoria allocata dinamicamente e invocare la funzione `deinit()` della coda;
- Ricordarsi di distinguere gli esempi nella descrizione dell'esercizio (che servono solo ad aiutare a comprendere il problema) dalle istruzioni di implementazione;
- È consentito definire ed implementare funzioni ausiliarie che possano aiutarvi nella soluzione del problema;
- 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 `iostream`.

coda.h
coda.cc
esercizio3.cc

Information for graders:

Total of marks: 40