FONDAMENTI DI PROGRAMMAZIONE
CORSO DI LAUREA IN INGEGNERIA INFORMATICA

PROVA PRATICA 16 SETTEMBRE 2019

Il tipo di dato astratto `Snake` rappresenta una versione semplificata del popolare videogioco presente in molti telefonini Nokia. Lo schema di gioco è composto da una griglia di R righe e C colonne. La casella in alto a sinistra nello schema ha indice (0, 0). All'interno dello schema, un serpente occupa un numero di caselle pari alla sua attuale lunghezza. Ogni casella occupata dal serpente viene identificata da un numero intero. In particolare, la casella occupata dalla testa del serpente viene indicata dal numero 1. Le caselle occupate dal corpo del serpente sono numerate in ordine crescente, fino ad arrivare alla coda del serpente. Implementare le seguenti operazioni che possono essere effettuate su uno `Snake`:

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

✓ **`Snake s(r, c);`**

Costruttore che inizializza uno `Snake` di `r` righe e `c` colonne. Il numero di righe e di colonne non può essere minore di 4. All'inizio, lo schema contiene un serpente lungo 4 caselle, allineato in verticale lungo la prima colonna a sinistra, con la coda giacente sulla casella in basso a sinistra e la testa verso nord.

✓ **`cout << s;`**

Operatore di uscita per il tipo `Snake`. L'output è nel formato rappresentato a destra.

Le cifre decimali rappresentano il serpente. Il numero 1 rappresenta la testa mentre il numero massimo (5 nell'esempio) rappresenta la coda. Le righe di caratteri `'-'` delimitano lo schema in alto ed in basso, le colonne di caratteri `'|'` lo delimitano a destra e a sinistra.

```
|--------|
|321     |
|4       |
|5       |
|        |
|        |
|        |
|--------|
```

✓ **`s.muovi(dir);`**

Operazione che muove la testa del serpente di una casella nella direzione specificata da `dir`, che può valere `'n'` (direzione nord), `'s'` (sud), `'o'` (ovest), `'e'` (est). Il resto del corpo segue la testa strisciando sullo schema. La casella dove si trovava la coda viene lasciata libera. Ad esempio, invocando `muovi('e')` sullo `Snake` di cui sopra, lo `Snake` risultante sarà quello rappresentato a destra.

```
|--------|
|4321    |
|5       |
|        |
|        |
|        |
|        |
|--------|
```

Il serpente non può uscire dallo schema e non può occupare con la testa una casella già occupata dal proprio corpo. Il serpente non può nemmeno invertire il proprio verso, cioè muoversi nella direzione opposta rispetto a quella in cui è rivolta la testa. Per esempio, nello `Snake` rappresentato a destra, il serpente non può muoversi verso ovest. In tutti i sopracitati casi, `muovi()` non fa niente e lo `Snake` rimane inalterato. Implementare l'operazione in modo che sia possibile concatenare più chiamate alla funzione, per esempio: `s.muovi('s').muovi('e').muovi('n').`

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

✓ **`s.mela(i, j)`**

Operazione che piazza una mela nella casella avente riga `i` e colonna `j`. Se la casella è già occupata da una mela o dal serpente, l'operazione non fa niente. L'operatore di output deve essere modificato per indicare con il carattere `'*'` una casella occupata da una mela, come nell'esempio a destra. La funzione `muovi()` deve essere modificata in modo tale che quando la testa del serpente occupa una casella occupata da una mela, la mela sparisce ed il corpo del serpente si allunga di una casella. L'esempio raffigurato a destra mostra il risultato della chiamata `muovi('n')`. In ogni caso, il serpente non può diventare più lungo di 9 caselle. Se un serpente di 9 caselle mangia una mela, la mela sparisce ma il suo corpo non si allunga. Implementare l'operazione in modo che sia possibile concatenare più chiamate alla funzione, per esempio: `s.mela(0,1).mela(3,3).mela(5,2).`

```
|--------|  |--------|
|    *   |  |    1   |
| 8  1   |  | 9  2   |
| 7  2   |  | 8  3   |
| 6543   |  | 7654   |
|        |  |        |
|        |  |        |
|--------|  |--------|
```

✓ **`s.inverti();`**

Operazione che inverte il verso del serpente, ovvero scambia la testa con la coda. Ad esempio, invocando l'operazione sullo `Snake` di cui sopra, l'output risultante sarà quello rappresentato a destra.

```
|--------|
|    9   |
| 1  8   |
| 2  7   |
| 3456   |
|        |
|        |
|--------|
```

✓ **`--s;`**

Operatore di pre-decremento per il tipo `Snake`, che accorcia il serpente di una casella dalla coda. In ogni caso, la lunghezza del serpente non può diventare minore di 4 caselle. Se l'operazione viene invocata su un serpente avente lunghezza di 4 caselle, lo `Snake` rimane inalterato.

✓ **`~Snake();`**

Distruttore.

Mediante il linguaggio C++, realizzare il tipo di dato astratto **`Snake`**, definito dalle precedenti specifiche. **Gestire le eventuali situazioni di errore**.

USCITA CHE DEVE PRODURRE IL PROGRAMMA

```
--- PRIMA PARTE ---
Test costruttore:
|--------|
|        |
|        |
|1       |
|2       |
|3       |
|4       |
|--------|

Test muovi:
|--------|
|        |
|321     |
|4       |
|        |
|        |
|        |
|--------|

|--------|
|  1     |
|432     |
|        |
|        |
|        |
|        |
|--------|

|--------|
| 12     |
| 43     |
|        |
|        |
|        |
|        |
|--------|

|--------|
| 12     |
| 43     |
|        |
|        |
|        |
|        |
|--------|

|--------|
|123     |
|  4     |
|        |
|        |
|        |
|        |
|--------|
```

```
--- SECONDA PARTE ---
Test mela:
|--------|
|123     |
|  4     |
|*       |
|*       |
|        |
|        |
|--------|

|--------|
|456     |
|3       |
|2       |
|1       |
|        |
|        |
|--------|

Test inverti:
|--------|
|321     |
|4       |
|5       |
|6       |
|        |
|        |
|--------|

Test operator--:
|--------|
|321     |
|4       |
|5       |
|        |
|        |
|        |
|--------|

Test distruttore:
(s1 e' stato distrutto)
```

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**.