Una `Battaglia` è una classe che rappresenta il gioco battaglia navale. Il gioco si svolge all'interno di un rettangolo di gioco di dimensione *m* x *m*. Partecipano al gioco *n* giocatori, ciascuno individuati da un intero strettamente positivo. A ciascuna casella del campo di gioco è assegnata una coordinata (*x*, *y*) (*x* ed *y* variano da 0 a *m*-1 e la casella (0,0) è quella in alto a sinistra). Ogni giocatore può aggiungere navi a forma rettangolare all'interno del rettangolo di gioco.

Internamente la battaglia navale è rappresentata attraverso una matrice quadrata di interi di dimensioni *m* x *m*. Ciascun elemento della matrice rappresenta una casella del campo di battaglia le cui coordinate (*x*, *y*) sono individuate dagli indici della posizione all'interno della matrice stessa. Il valore intero di ogni elemento della matrice rappresenta lo stato della casella del rettangolo di gioco. Nello specifico: il valore 0 indica una casella vuota non occupata da nessuna nave, un valore strettamente positivo rappresenta una parte di una nave posizionata in quella casella che ancora non è stata colpita, un valore negativo rappresenta invece una nave che è stata colpita. Il valore assoluto dell'intero viene usato per memorizzare l'identificativo del giocatore al quale appartiene la nave.

Le seguenti operazioni possono essere compiute su di una `Battaglia`:

`--- PRIMA PARTE ---` *(qualora siano presenti errori di compilazione, collegamento o esecuzione in questa parte,* ***l'intera prova sarà considerata insufficiente*** *e pertanto* ***non sarà corretta****)*

- ✓ **`Battaglia b(m, n);`**

Costruttore che crea una `Battaglia` di dimensioni `m` per `m` con un numero di giocatori pari a `n`. Tutti gli elementi della matrice vengono inizializzati a 0 (non ci sono navi).

- ✓ **`b.aggiungi(x1, y1, x2, y2, id);`**

Operazione che aggiunge una nave di forma rettangolare all'interno della battaglia `b`. La nave viene posizionata a partire dalla casella di coordinate (x1, y1) fino a quella di coordinate (x2, y2), purchè le coordinate passate alla funzione siano valide e rispettino le condizioni: $x1 \leq x2$, $y1 \leq y2$. La funzione ha successo (ed in tale caso restituisce `true`) se e solo se le caselle che cadono del rettangolo (x1, y1)(x2, y2) sono tutte vuote e l'`id` del giocatore è valido.

La funzione fallisce (restituendo `false`) tutte le volte in cui nel rettangolo (x1, y1)(x2, y2) si trovi una parte o l'intera nave dello stesso o di altro giocatore (che sia stata o meno colpita non fa alcuna differenza).

Esempio:

```
0 0 0 0
0 2 2 2   battaglia a seguito di aggiungi(1,1,2,3,2)
0 2 2 2
0 0 0 0
```

- ✓ **`cout << b;`**

Operatore di uscita per il tipo `Battaglia`. L'uscita ha il seguente formato:

```
1 1 1 0
0 2 2 2
1 2 2 2
1 0 0 0
```

L'output mostrato corrisponde a una `Battaglia` 4x4 con 2 giocatori.

✓ **`b.fuoco(x,y);`**

Operazione che restituisce `true` se una nave in posizione (x,y) viene colpita e restituisce `false` in caso contrario. La funzione deve aggiornare in maniera consistente i valori della matrice

Esempio: Nel caso in cui m sia la seguente, la `b.fuoco(2,2)` restituirà `false`, mentre la `b.fuoco(0,0)` restituirà `true` modificando la matrice come segue:

```
-1 1 1 0
0 2 0 0
1 2 0 0
1 2 0 0
```

--- SECONDA PARTE ---

✓ **`b += b1;`**

Operazione che modifica il campo di battaglia di `b` unendoci quello di `b1`. Le due battaglie devono avere la stessa dimensione e lo stesso numero di giocatori. Nel caso in cui almeno un conflitto è presente, cioè una nave del campo `b1` è in sovrapposizione (anche parziale) con una nave di `b`, l'unione viene annullata e lo stato di `b` rimane inalterato. Nel caso l'operazione vada a buon fine la funzione restituisce `true, false` altrimenti.

✓ **`b == id`**

Operatore di confronto (*doppio uguale*), che restituisce `true` se il giocatore `id` è ancora in gioco nella battaglia `b`, `false` altrimenti.

✓ **`~Battaglia()`**

Distruttore.

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

USCITA CHE DEVE PRODURRE IL PROGRAMMA

```
--- PRIMA PARTE ---
1 1 0 0
0 2 2 2
1 2 2 2
1 0 0 0

1 1 0 0
0 2 2 2
1 2 2 2
1 0 5 5

Nave colpita
Colpo a vuoto
Nave colpita

1 1 0 0
0 2 2 -2
1 2 2 2
1 0 5 -5
```

```
--- SECONDA PARTE ---
1 1 1 1 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0

0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
2 2 2 2 0 0
1 1 1 1 0 0

Unione avvenuta
1 1 1 1 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
2 2 2 2 0 0
1 1 1 1 0 0

1 1 1 1 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
-2 -2 -2 -2 0 0
1 1 1 1 0 0

Giocatore 1 ancora in gioco
Giocatore 2 eliminato
```