Un `Bit16` è un tipo di dato astratto che modella un vettore di 16 bit. Ad ogni bit è associata una posizione: il bit di posizione 0 è quello più a destra, e quello di posizione 15 quello più a sinistra. Un oggetto di tipo `Bit16` deve essere mostrato a video nel seguente modo:

```
+----------------+
|FEDCBA9876543210|
+----------------+
|0111111111111110|
+----------------+
```

Nel caso in esempio, l'oggetto ha i bit di posizione 15 e quello di posizione 0 a 0, e tutti gli altri ad 1. Il valore del singolo bit (0 o 1) viene mostrato nella riga sottostante, mentre la posizione nella riga sovrastante. La posizione è indicata con i caratteri '0', '1', ... , '9' per le posizioni da 0 a 9, e con i caratteri 'A', ..., 'F' per le posizioni da 10 a 15.

Il tipo di dato astratto `Mask16` modella invece una maschera di ampiezza 16. La maschera può essere di uni oppure di zeri. Un oggetto di tipo `Mask16` deve essere mostrato a video nel seguente modo:

```
+----------------+
|FEDCBA9876543210|
+----------------+
|##########1111##|
+----------------+
```

qualora si tratti di una maschera di *uni* (detta anche nel seguito 1-maschera), e nel seguente modo:

```
+----------------+
|FEDCBA9876543210|
+----------------+
|0000########0000|
+----------------+
```

qualora si tratti di una maschera di *zeri* (detta anche nel seguito 0-maschera).

La 1-maschera del primo esempio verrà utilizzata per agire solo sui bit di posizione 2, 3, 4, e 5, mentre la 0-maschera del secondo esempio verrà utilizzata per agire solo sui primi ed ultimi 4 bit. Vedremo nel seguito che per modificare in maniera selettiva alcuni bit di un oggetto `Bit16` converrà utilizzare una 1-maschera, mentre in altre situazione converrà utilizzare una 0-maschera.

Implementare le seguenti operazioni, che operino su `Bit16` e/o su `Mask16`.

NB: È ASSOLUTAMENTE OBBLIGATORIO MEMORIZZARE I BIT IN MODO COMPATTO, ovvero un bit in memoria per ogni bit rappresentato. Altrimenti la valutazione verrà decurtata di molti punti

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

- ✓ **`Bit16 b(ui);`**

Costruttore che inizializza l'oggetto `b` mediante l'intero senza segno `ui` passato come argomento. Ad esempio, nel caso venga invocato con l'argomento `10u` (il naturale dieci), il contenuto di `b` dovrà essere:

```
+----------------+
|FEDCBA9876543210|
+----------------+
|0000000000001010|
+----------------+
```

ossia dovrà coincidere con la codifica base 2 dell'intero senza segno specificato.

✓ **`cout << b;`**

Operatore di uscita per l'oggetto `Bit16 b`, il cui comportamento è stato già illustrato in precedenza.

✓ **`Mask16 m(ui, unoMaschera);`**

Costruttore per la classe `Mask16`, che prende in ingresso l'intero senza segno `ui` e il booleano `unoMaschera`. Se `unoMaschera` vale `true`, allora `m` sarà una maschera di *uni*, altrimenti sarà una maschera di *zeri*.

Esempi:

```
+----------------+
|FEDCBA9876543210|
+----------------+
|############1#1#|
+----------------+
```

Maschera di *uni* relativa alla chiamata al costruttore con argomenti `10` e `true`, rispettivamente.

```
+----------------+
|FEDCBA9876543210|
+----------------+
|###########0#0#0|
+----------------+
```

Maschera di *uni* relativa alla chiamata al costruttore con argomenti `21` e `false`, rispettivamente.

In altre parole, e come vedremo più avanti, il senza segno `ui` specifica su quali bit di un oggetto `Bit16` dovrà agire la 1-maschera (o la 0-maschera, a seconda dei casi). Sugli altri bit la maschera **non dovrà agire**. Quando una maschera viene mostrata a video, i bit sui quali non agirà verranno mostrati con il simbolo '#'.

✓ **`cout << m;`**

Operatore di uscita per un oggetto `Mask16`, il cui comportamento è stato già illustrato in precedenza. Si rimarca il fatto che i bit sui quali la maschera non opererà debbono apparire con il simbolo '#', mentre quelli sui quali la maschera opererà debbono essere degli 0 (nel caso di 0-maschera) o degli 1 (nel caso di 1-maschera).

✓ **`b|m;`**

Operatore di or bit-a-bit, definito tra un oggetto `Bit16` (primo operando) ed uno `Mask16` (secondo operando). L'operazione deve resituire un nuovo oggetto di tipo `Bit16`, avente in posizione *i*-esima l'or bit-a-bit tra i corrispondenti bit di `b` e `m`, solo per i bit sui quali la maschera `m` è attiva, e deve lasciare gli altri identici a quelli presenti in `b`, laddove la maschera non è attiva. Il primo operando `b` **deve rimanere inalterato**.

Ad esempio, dato il seguente oggetto `Bit16`:

```
+----------------+
|FEDCBA9876543210|
+----------------+
|1010010000100101|
+----------------+
```

ed il seguente oggetto `Mask16` (si tratta di una 1-maschera, in questo caso):

```
+----------------+
|FEDCBA9876543210|
+----------------+
|##########1111##|
+----------------+
```

l'operatore deve restituire il seguiente oggetto `Bit16`:

```
+----------------+
|FEDCBA9876543210|
+----------------+
|1010010000111101|
+----------------+
```

Come si può notare, l'applicazione della maschera forza i bit da 2 a 5 (evidenziati in grigio) ad essere ad 1. Si noti come l'or bit-a-bit con una 0-maschera non alteri nessun bit, e pertanto il risultato sarà una copia di `b`.

✓ **`b&m;`**

Operatore di and bit-a-bit, definito tra un oggetto `Bit16` (primo operando) ed uno `Mask16` (secondo operando). L'operazione deve restituire un nuovo oggetto di tipo `Bit16`, avente in posizione *i*-esima l'and bit-a-bit tra i corrispondenti bit di `b` e `m`, solo per i bit sui quali la maschera `m` è attiva, e deve lasciare gli altri identici a quelli presenti in `b`, laddove la maschera non è attiva. Si noti come l'applicazione di una 0-maschera porterà i bit sui quali la maschera è attiva a 0, lasciando gli altri inalterati. Si noti inoltre che l'and bit-a-bit con una 1-maschera deve restituire una copia del primo operando, ossia non altera alcun bit.

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

✓ **`b^m;`**

Operazione che calcola l'or esclusivo bit-a-bit tra l'oggetto `b` di tipo `Bit16` e l'oggetto `m` di tipo `Mask16` e restituisce uno nuovo oggetto di tibo `Bit16`. Qualora la maschera `m` sia una 1-maschera, l'*i*-esimo bit di `b` deve essere invertito se la maschera è attiva per quel bit, e rimanere inalterato altrimenti.

Ad esempio, supponendo che l'oggetto `b` sia il seguente:

```
+----------------+
|FEDCBA9876543210|
+----------------+
|1010010000100101|
+----------------+
```

e la maschera m sia la seguente (1-maschera):

```
+----------------+
|FEDCBA9876543210|
+----------------+
|##########1111##|
+----------------+
```

allora l'oggetto `Bit16` risultato sarà il seguente:

```
+----------------+
|FEDCBA9876543210|
+----------------+
|1010010000011001|
+----------------+
```

NB: Questa operazione (e questa soltanto) deve essere implementata in maniera tale che sia possibile effettuare anche `m|b` (il risultato sarà un `Bit16` anche in questo caso), ossia l'operatore deve essere simmetrico.

✓ **`m.quantiMascherati();`**

Operazione che agisce su una maschera e che restituisce il numero di bit mascherati (ossia quanti '#' contiene al suo interno).

✓ **`m.ruotaADestra(quanti);`**

Operazione che ruota a destra **ed in maniera circolare** una maschera, di un numero di posizioni pari a `quanti`.

✓ **`b.mascheraConvertitrice(bArrivo)`**

Operazione che, dato un oggetto `b` di tipo `Bit16` ed un secondo oggetto `bArrivo`, dello stesso tipo, restituisce come risultato un nuovo oggetto di tipo `Mask16`. La maschera risultato deve essere quella 1-maschera tale che, una volta applicata a `b` in or esclusivo bit-a-bit, restituisce come risultato proprio un oggetto identico a `bArrivo`.

Mediante il linguaggio C++, realizzare i tipi di dato astratto `Bit16` e `Mask16`, definiti dalle precedenti specifiche. Non è permesso utilizzare funzionalità della libreria STL come il tipo `string`, il tipo `vector`, il tipo `list`, ecc. **Gestire le eventuali situazioni di errore**.

USCITA CHE DEVE PRODURRE IL PROGRAMMA

```
--- PRIMA PARTE ---

b
+----------------+
|FEDCBA9876543210|
+----------------+
|1010010000100101|
+----------------+


b2
+----------------+
|FEDCBA9876543210|
+----------------+
|0111111111111110|
+----------------+


m1
+----------------+
|FEDCBA9876543210|
+----------------+
|##########1111##|
+----------------+


m0
+----------------+
|FEDCBA9876543210|
+----------------+
|##########0000##|
+----------------+


b | m1 (mette ad 1 i bit di b specificati dalla 1-maschera m1)
+----------------+
|FEDCBA9876543210|
+----------------+
|1010010000111101|
+----------------+


b & m0 (mette ad 0 i bit di b specificati dalla 0-maschera m0)
+----------------+
|FEDCBA9876543210|
+----------------+
|1010010000000001|
+----------------+


b & m1 (mascheramento inutile: stampa copia di b)
+----------------+
|FEDCBA9876543210|
+----------------+
|1010010000100101|
+----------------+


b | m0 (mascheramento inutile: stampa copia di b)
+----------------+
|FEDCBA9876543210|
+----------------+
|1010010000100101|
+----------------+
```

```
--- SECONDA PARTE ---

b ^ m1 (inverte i bit di b specificati dalla 1-maschera m1)
+----------------+
|FEDCBA9876543210|
+----------------+
|1010010000011001|
+----------------+


b ^ m0 (mascheramento inutile: stampa copia di b)
+----------------+
|FEDCBA9876543210|
+----------------+
|1010010000100101|
+----------------+


Il numero di bit mascherati dalla maschera m0 e' (deve stampare 12)
12
Il numero di bit mascherati dalla maschera m1 e' (deve stampare 12)
12

Ruoto a destra la maschera m1 di 4 (deve visualizzare 11############11)
+----------------+
|FEDCBA9876543210|
+----------------+
|11############11|
+----------------+


Test della funzione mascheraConvertitrice()
+----------------+
|FEDCBA9876543210|
+----------------+
|#1#11#11##1##1#1|
+----------------+
```

Regole riguardanti l'autocorrezione

Nei prossimi giorni vi verrà richiesto di effettuare l'autocorrezione. L'autocorrezione dovrete effettuarla tenendo presente che deve il vostro elaborato **dovrà** riprodurre l'uscita corretta **sia della prima che della seconda parte.**

In altre parole, i docenti non correggeranno a loro volta il vostro elaborato se:

1) l'uscita della prima parte alla consegna era scorretta
2) l'uscita di prima e seconda parte, dopo la vostra autocorrezione, non è corretta.

NB: Ovviamente le correzioni che apporterete non vi consentiranno di superare la prova grazie ad un ravvedimento "fuori tempo massimo". Servono solo a velocizzare l'operazione di correzione da parte dei docenti. Inoltre è un utile esercizio didattico, in cui lo studente può prendere coscienza dei propri errori e, in certi casi, dimostrare che con poche modifiche/intergazioni, anche l'uscita della seconda parte sarebbe stata corretta.